1999-03

Hardware integration of the Small Autonomous Underwater Vehicle Navigation System (SANS) using a PC/104 computer

Akyol, Kadir

Monterey, California: Naval Postgraduate School

http://hdl.handle.net/10945/13575

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

HARDWARE INTEGRATION OF THE SMALL AUTONOMOUS UNDERWATER VEHICLE NAVIGATION SYSTEM (SANS) USING A PC/104 COMPUTER

by

Kadir Akyol

March 1999

Thesis Advisor:	Xiaoping Yun
Co-Advisor:	Eric R. Bachmann

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY ***(Leave blank)***

2. REPORT DATE
March 1999

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
HARDWARE INTEGRATION OF THE SMALL AUTONOMOUS UNDERWATER VEHICLE NAVIGATION SYSTEM (SANS) USING A PC/104 COMPUTER

5. FUNDING NUMBERS

6. AUTHOR(S) Akyol, Kadir

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(maximum 200 words)***

At the Naval Postgraduate School (NPS), a small AUV navigation system (SANS) has been developed for research in support of shallow-water mine countermeasures and coastal environmental monitoring. The objective of this thesis is to develop a new version of SANS, aimed at reducing size and increasing reliability by utilizing state-of-the-art hardware components.

The new hardware configuration uses a PC/104 computer system, and a Crossbow DMU-VG Six-Axis Inertial Measurement Unit (IMU). The PC/104 computer provides more computing power and more importantly, increases the reliability and compatibility of the system. Replacing the old IMU with a Crossbow IMU eliminates the need for an analog-to-digital (A/D) converter, and thus reduces the overall size of the SANS.

The new hardware components are integrated into a working system. A software interface is developed for each component. An asynchronous Kalman filter is implemented in the current SANS system as a navigation filter. Bench testing is conducted and indicates that the system works properly. The new components reduce the size of the system by 52% and increase the sampling rate to more than 80Hz.

14. SUBJECT TERMS
INS, GPS, AUV,SANS, Navigation, Kalman Filter

14. NUMBER OF PAGES
165

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFI- CATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT
ULd

Approved for public release; distribution is unlimited

HARDWARE INTEGRATION OF THE SMALL AUTONOMOUS UNDERWATER VEHICLE NAVIGATION SYSTEM (SANS) USING A PC/104 COMPUTER

Kadir Akyol
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1999

Author: [redacted]
Kadir Akyol

Approved by: [redacted]
Xiaoping Yun, Thesis Advisor

[redacted]
Eric R. Bachmann, Co-Advisor

[redacted]
Jeffrey B. Knorr, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

At the Naval Postgraduate School (NPS), a small AUV navigation system (SANS) has been developed for research in support of shallow-water mine countermeasures and coastal environmental monitoring. The objective of this thesis is to develop a new version of SANS, aimed at reducing size and increasing reliability by utilizing state-of-the-art hardware components.

The new hardware configuration uses a PC/104 computer system, and a Crossbow DMU-VG Six-Axis Inertial Measurement Unit (IMU). The PC/104 computer provides more computing power and more importantly, increases the reliability and compatibility of the system. Replacing the old IMU with a Crossbow IMU eliminates the need for an analog-to-digital (A/D) converter, and thus reduces the overall size of the SANS.

The new hardware components are integrated into a working system. A software interface is developed for each component. An asynchronous Kalman filter is implemented in the current SANS system as a navigation filter. Bench testing is conducted and indicates that the system works properly. The new components reduce the size of the system by 52% and increase the sampling rate to more than 80Hz.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my sincerest thanks to Dr. Xiaoping Yun for guiding me through this enlightening experience. His patience and understanding gave me the motivation to keep moving forward and use common sense. I am much appreciative of all the support he has given to my efforts on the SANS project. I also wish to give thanks to Dr. Robert McGhee and Eric Bachmann for their help during this research. Eric Bachmann was always ready for questions and never hesitated to answer. I would like to thank him for his friendly assistance.

DEDICATION

For my mother and wife

I. INTRODUCTION

A. BACKGROUND

An Autonomous Underwater Vehicle (AUV) can be capable of numerous missions, both overt and clandestine. Such vehicles have been used for inspection, mine countermeasures, survey, and observation [Ref. 1]. One of the most important and difficult aspects of an AUV mission is navigation. In order to achieve a wide variety of missions, the navigation system of the AUV must be accurate. AUV navigation may be accomplished using the Global Positioning System (GPS) and an Inertial Navigation System (INS). GPS is capable of supplying accurate navigation, if it is integrated with INS to compensate for the loss of GPS signals due to environmental blockages. GPS provides accurate positioning when the AUV is surfaced, while the INS is used for submerged navigation.

At the Naval Postgraduate School (NPS), a Small AUV Navigation System (SANS) has been developed for research in support of shallow-water mine countermeasures and coastal environmental monitoring [Ref. 2]. The goal of designing this system is to demonstrate the feasibility of using a small, low-cost Inertial Measurement Unit (IMU) to navigate between Differential Global Positioning System (DGPS) fixes.

The current version of SANS is composed of "off the shelf " components, which include an IMU, GPS/DGPS Receiver, magnetic compass, water speed sensor, and data processing computer.

An asynchronous Kalman Filter, which has six states for orientation estimation, and eight states for position estimation, is used in the system as navigation software. The SANS system has been upgraded using an AMD 586DX133 based PC/104 computer to provide more computing power and more importantly to increase reliability and provide compatibility with PC/104 industrial standards. [Ref. 3]

The goal of this thesis is to contribute to the ongoing AUV research project at NPS by integrating hardware and software of the INS/GPS navigation system using an AMD 586DX133 based PC/104 computer.

B. RESEARCH QUESTIONS

This thesis will examine the following research areas:

- Integrate an AMD 586DX133 based PC/104 computer into the SANS system.

- Develop the software interface, which will communicate between GPS receiver, compass, IMU and, processing computer.

- Implement the asynchronous Kalman filter developed by reference [3] into the new hardware system.

- Test and evaluate the hardware and software.

C. SCOPE, LIMITATIONS, AND ASSUMPTIONS

This thesis reports part of the findings of more than seven years of research in an ongoing project. The main scope of this thesis is to integrate an AMD 586DX133 based PC/104 computer into the SANS system to provide more computing power and, more importantly, to increase reliability and computability.

D. ORGANIZATION OF THESIS

Chapter II provides a detailed description of the hardware components and system integration.

Chapter III describes the software additions, and changes to support current hardware configurations.

Chapter IV presents testing results of the system.

Chapter V presents the thesis conclusions and provides recommendations for future research.

II. SANS HARDWARE CONFIGURATION

A. INTRODUCTION

Previous versions of the SANS hardware configuration are described in References [4,5,6,7, and 8]. The purpose of this chapter is to present detailed information about the current SANS hardware configuration and operation. Most of the previous SANS hardware parts have been replaced with more powerful, more flexible, and more reliable components which are faster, smaller, and cheaper.

Figure 2.1 shows the current SANS hardware configuration. The main hardware components are the Crossbow DMU-VG Six Axis IMU, the Oncore GPS/DGPS Receiver, the Precision Navigation TCM2 Electronic Compass, the SonTek Hydra Water Speed Sensor, the Sealevel C4-104 Serial I/O Module, and the AMD 586DX133 based PC/104.

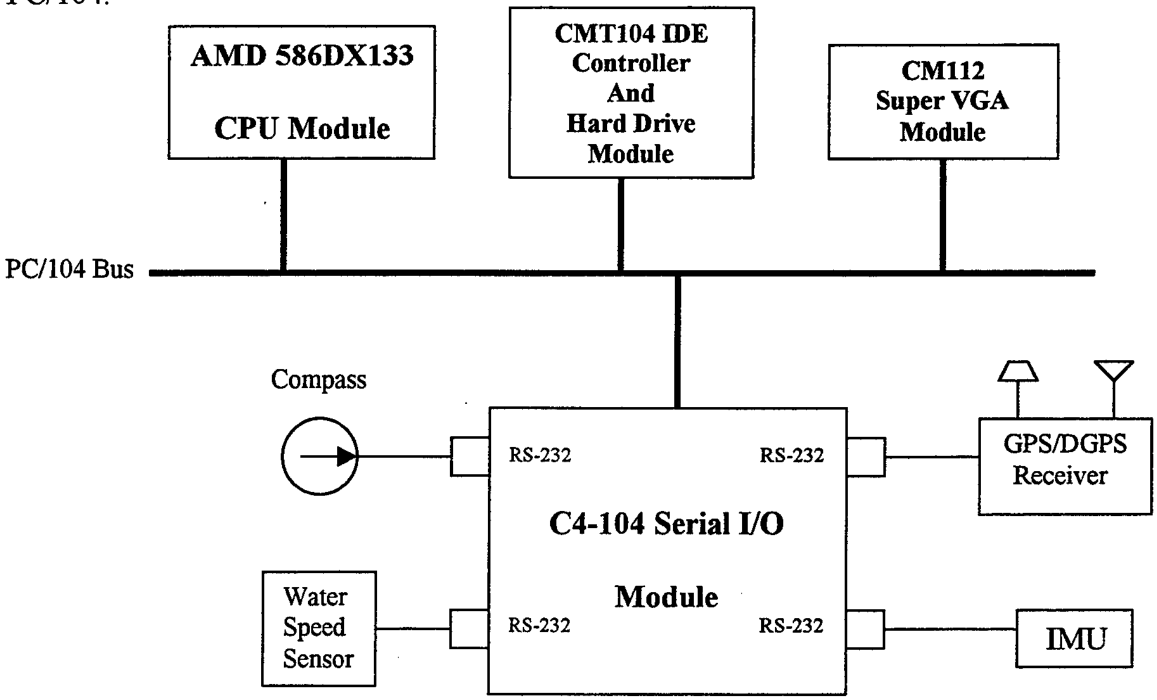

Figure 2.1: Current SANS Hardware Configuration

B. HARDWARE DESCRIPTION

1. Precision Navigation TCM2 Electronic Compass

The Precision Navigation model TCM2 Electronic Compass Module is used in the current SANS hardware configuration. The TCM2 consists of a three-axis magnetometer, a two-axis tilt sensor and a small A/D board. Output includes roll, pitch, heading, and a three dimensional magnetic field measurement. It is accurate to within one half of a degree in level operation. The TCM2 will provide more accurate heading information following calibration (performed by user) for local magnetic field distortions. It provides an alarm when local magnetic anomalies are present, and gives out-of-range warnings when the unit is being tilted too far. The calibration of the compass and its error characteristics are described in [Ref. 6]. It requires 5 VDC and 15-22 mA [Ref. 9].

2. Real Time Devices AMD 586DX133 Based PC/104

The Real Time Devices AMD 586DX133 based PC/104 is employed as a data acquisition and processing unit in the current version of the SANS system. PC/104 is an industrial standard for PC-compatible modules that can be stacked together to create embedded computer applications. This system fulfills the basic needs of embedded systems such as low power consumption, modularity, small foot print, high reliability, good noise immunity, high speed operation, and expandability.

The PC/104 can be easily customized by stacking PC/104 modules that are compliant with the PC/104 bus architecture, such as video controllers, network interfaces, analog and digital data acquisition modules, sound I/O modules etc.

The SANS system is equipped with four PC/104 modules. These are the AMD 586DX133 CPU Module, the CMT104 IDE Controller and Hard Drive Module, the CM112 Super VGA Module, and the C4-104 Serial I/O Module.

The PC/104 CPU module offers all major functions of a standard PC computer on one compact board. Figure 2.2 shows a simplified block diagram of the PC/104 CPU module. It has all primary I/O functions of a standard PC computer including a keyboard interface, a parallel port, two serial ports, a Real Time Clock, and a speaker port. It also enhances standard PC compatible computer systems by adding: Solid State Disk sockets, a non-volatile configuration EEPROM, and a Watchdog Timer [Ref. 10].

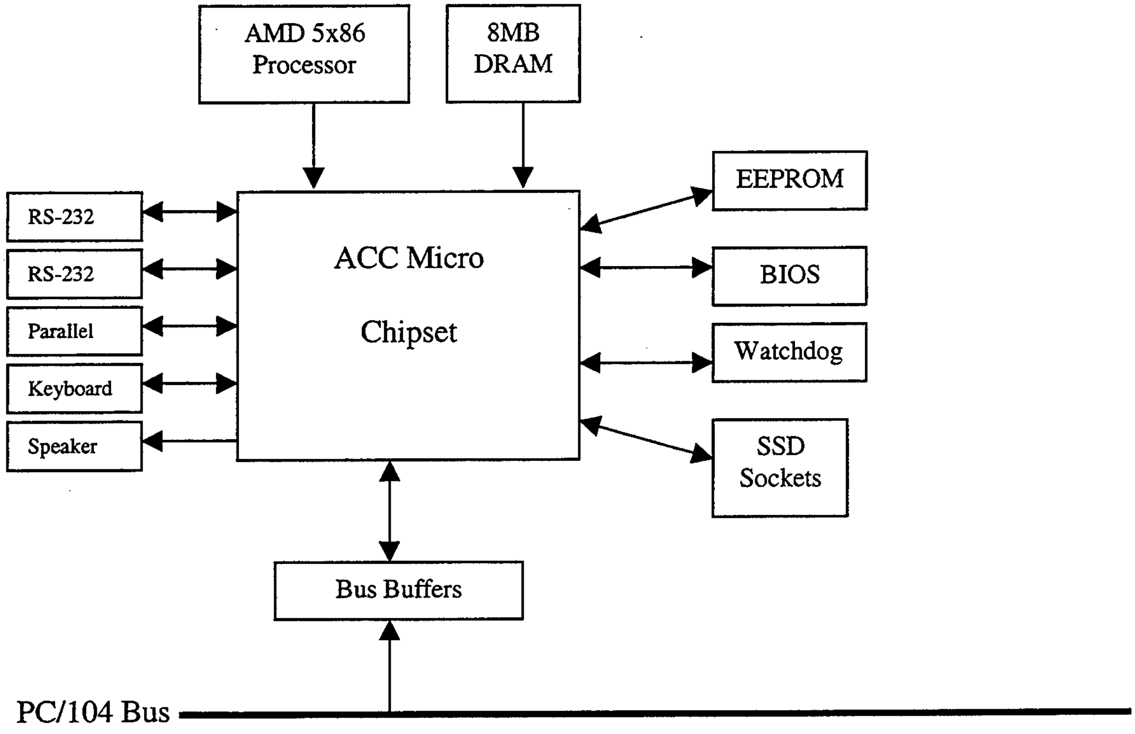

Figure 2.2: PC/104 CPU Module

The CMT104 IDE Controller and Hard Drive Module were designed to integrate IDE hard drive or Flash Drive in the PC/104 stack to support the CPU module. It allows up to four drives in the system. [Ref. 11]

The CM112 Super VGA Module was designed to provide Super VGA video, as well as floppy and hard drive support for the CPU module. Super VGA has a resolution of up to 1024 x 768 pixels with at least 256 colors. [Ref. 12]

The PC/104 is implemented with PC compatible BIOS, which supports the ROM-DOS and MS-DOS operating systems. Drivers in the BIOS can boot from the floppy disk, hard disk, Solid State Disk (SSD), or a serial port link. [Ref. 10]

The system uses on AMD Am5x86 microprocessor with 133Mhz. clock speed. It's physical dimensions are 3.6 x 3.8 x 0.6 inches (97 x 100 x 16 mm) and its weight is 3.4 ounces (100 grams). It operates on 5 VDC +/- 5%. Power consumption depends on the peripherals connected to the board, the selected SSD configuration, and the memory configuration. The power consumption for typical configurations is 1.75 A (8.75 W). The PC/104 CPU module has a 12MB (expandable to 72MB) disk on chip that will store the SANS code and any other data it uses. An integral Viper 170MB hard drive on the CMT104 Hard Drive Module is also available for more data storage. [Ref. 10]

3. Sealevel C4-104 Serial I/O Module

The SANS system uses four serial port connections for the sensors. These are the IMU, DGPS, compass, and water speed sensor. Two serial ports usually come standard on PCs. The Sealevel C4-104 serial I/O module provides four RS-232 serial I/O ports for the PC/104 application. Each serial port has its own base memory addresses and

Interrupt Request (IRQ) assignments. Table 2.1 shows the base address and IRQ setting used with SANS. The address and IRQ selection options are described in Reference [13].

	Base Address (hex.)	IRQ
Port 1	3F8	4
Port 2	2F8	3
Port 3	3E8	5
Port 4	2E8	3

Table 2.1: The base address and IRQ settings [Ref. 13]

The C4-104 is compliant with PC/104 specification including both mechanical and electrical specifications. The C4-104 utilizes 6554 Universal Asynchronous Receiver/Transmitters (UART) with programmable baud rates, data format, interrupt control and a 16-byte input and output FIFO. The system operates on 5 volt DC. [Ref. 13]

4. Crossbow DMU-VG Six Axis Inertial Measurement Unit

The DMU-VG (Figure 2.3) is a six-axis measurement system designed to measure linear acceleration along three orthogonal axes, and rotation rates around three orthogonal axes. It is designed to provide stabilized pitch and roll in dynamic environments [Ref. 14]. The IMU has both analog output and RS-232 serial port output. Previous versions of SANS were equipped with a Systron Donner MotionPak IMU, which delivered data only in analog format. Thus, replacing the MotionPak IMU with the Crossbow DMU-VG IMU eliminated the need for a PC/104 A/D module, and therefore reduced the total size of the SANS unit. The general specifications of the DMU-VG are shown in Table

2.2 and the structure of the data packet sent over the RS-232 interface is shown in Table 2.3.

Chapter III presents more detailed information about Crossbow DMU-VG IMU interface.

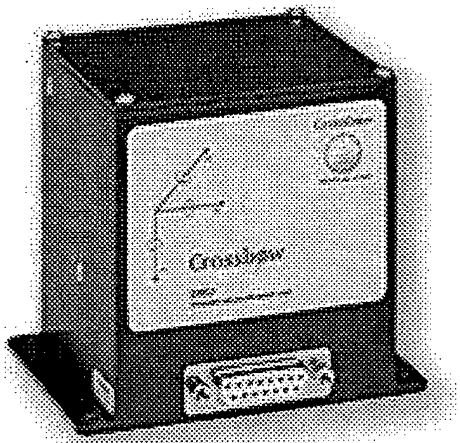

Figure 2.3: Crossbow DMU-VG [Ref. 14]

Parameter	Units	Range
Roll Range	deg.	-/+ 180
Pitch Range	deg.	-/+ 90
Roll, Pitch Angle: Dynamic Accuracy	deg. RMS	1
Roll, Pitch Angle: Repeatability	deg.	0.5
Roll, Pitch, Yaw Angular Rate Resolution	deg./sec.	0.05
Bandwidth	Hz.	10
Input Supply Voltage	Volt. DC.	8 – 30
Input Supply Current	mA(max.)	100
Package	Inches	3 x 3.375 x 3.250
Weight	Grams	475
Operating Temperature Range	degrees C.	40 to 85

Table 2.2: Crossbow IMU Specifications [Ref. 15]

Note: Most Significant Bit (MSB) Least Significant Bit (LSB)

Byte	D M U – V G
0	Header (255)
1	Roll (MSB)
2	Roll (LSB)
3	Pitch (MSB)
4	Pitch (LSB)
5	Roll Rate X (MSB)
6	Roll Rate X (LSB)
7	Pitch Rate (MSB)
8	Pitch Rate Y (LSB)
9	Yaw Rate Z (MSB)
10	Yaw Rate (LSB)
11	Acceleration X (MSB)
12	Acceleration X (LSB)
13	Acceleration Y (MSB)
14	Acceleration Y (LSB)
15	Acceleration Z (MSB)
16	Acceleration Z (LSB)
17	Temp Sensor Voltage (MSB)
18	Temp Sensor Voltage (LSB)
19	Time (MSB)
20	Time (LSB)
21	Checksum

Table 2.3: Crossbow IMU Data Packet Format [Ref. 15]

5. Motorola Oncore GPS/DGPS Receiver

The GPS receiver used in the SANS system is the Motorola ONCORE Receiver. It is capable of tracking eight satellites simultaneously. The GPS receiver incorporates a DGPS capability. It operates on a 5 volt DC regulated power source. It's data port interface is RS-232 compatible. The output message consists of latitude, longitude, height, velocity, heading, time, and satellite tracking status. It can provide position accuracy of better than 25 meters Spherical Error Probable (SEP) with Selective

Availability (SA) and 100 meters SEP without SA. The typical Time-To-First-Fix (TTFF) is 18 seconds with a 2.5 second reacquisition time [Ref. 16].

6. SonTek Hydra Water Speed Sensor

The SonTek Hydra Water Speed Sensor is a single point, high resolution, 3D Doppler current meter. It measures the velocity of water using the Doppler effect. The device uses one transmitter and 3 acoustic receivers, which are aligned to intersect with the transmitted beam pattern. The velocity measured by each receiver is referred to as the bistatic velocity, and is the projection of the 3D velocity vector onto the bistatic axis of the acoustic receiver. The bistatic velocities are converted to XYZ velocities. The velocity data can be reported as the data in an Earth (North-East) fixed coordinate system by using compass and tilt sensors. The sensor provides data over a RS-232 serial port interface. [Ref. 17]

The general specifications of the system are presented in Table 2.4.

Parameter	Units	Range
Acoustic frequency	MHz.	5
Range (programmable)	cm/sec.	-/+ 5, 20, 50, 200, 500
Velocity resolution	mm/sec.	0.1
Sampling rate	Hz.	0.1 – 25
Internal recorder	Mbytes	20
Operating temperature	deg. C	-2 – 40
Power supply	Volt DC.	12 – 24
Power consumption	W	2.5 – 5
Max. deployment depth (Delrin)	m	250
Max. deployment depth (Stainless steel)	m	2000

Table 2.4: Hydra Water Speed Sensor Specifications [Ref. 17]

C. FUTURE COMPONENTS

For future SANS hardware configurations, a Local Area Network (LAN) connection to transmit data from SANS to a host processor is desired. In order to meet this need, the addition of Proxim Range LAN2 PC card to the system is being considered. The Range LAN2 system is capable of transmitting data at 1.6Mbps. through a PCMCIA card format at distances up to 1000 feet [Ref. 18]. This card provides the ability to observe data received from the sensors remotely.

The technological advances in GPS area make it possible change out the four year old DGPS package. Two possible products, Rockwell Semiconductor's NAVCAR LP, and Trimble's Pathfinder with ASPEN software, are being considered. With this feature, more accurate navigation information could be acquired, as well as smaller size advantages.

D. SUMMARY

The components in the current SANS hardware configuration were chosen based on size, cost, power, and ease of operation. The current SANS configuration uses a Crossbow DMU-VG Six Axis IMU, a C4-104 Serial I/O Module, and an AMD 586DX133 based PC/104 computer.

The new components reduced the size of the system by 52% [Ref. 3]. The new IMU unit provided both analog and digital data output. Thus, the need for a PC/104 A/D module was eliminated. The C4-104 Serial I/O Module provided four RS-232 serial ports for the PC/104 application. The AMD 586DX133 based PC/104 computer provided more computing power and, more importantly, increased reliability and compatibility with PC/104 industrial standards [Ref. 3]. Testing and the evaluation of the new SANS hardware configuration is currently in progress.

III. SOFTWARE DEVELOPMENT

A. INTRODUCTION

The purpose of the SANS software is to utilize IMU, heading, and water-speed information to implement an INS based on an asynchronous Kalman filter. The INS information is integrated with GPS information to obtain continuously accurate navigation information in real time.

Changes in the SANS hardware configuration have driven subsequent changes in the software design. The previous version of SANS used two serial ports to obtain data using a RS-232 interface. The GPS data was received via the COM1 serial port, and the compass data was received via the COM2 serial port. The previous IMU sensor provided data in analog format. Additional code was required in the SANS software to operate the A/D converter module and buffer this data.

In the current version of SANS, a Sealevel C4-104 module provides four serial port connections. The previous IMU has been replaced with a Crossbow DMU-VG six-axis measurement unit, which outputs digital data over RS-232. Thus, the software developed for the current version of SANS has four serial port data communication objects to acquire data from sensors. COM1 is assigned to the GPS, COM2 is assigned to the compass, COM3 is assigned to the IMU, and COM4 is assigned to the water speed sensor.

The constant-gain filter used in the previous version of SANS was replaced with an asynchronous Kalman filter, which has six states for orientation estimation, and eight states for position estimation.

The software was implemented in C^{++} and compiled using the Borland 5.0 compiler. It is designed to run on a standard DOS platform for use on an AMD586DX/133 MHz processor.

B. SOFTWARE DESCRIPTION

The current implementation of the SANS software continues to be based on the software described in reference [4]. The software changes and additions to support current SANS hardware are introduced in this chapter. Figure 3.1 shows the SANS software objects and data flow. Source code for these objects can be found in Appendix A and B.

1. GPS Data

The GPS class object obtains GPS position messages in the Motorola 8-Channel Position/Status/Data Output Message (@@Ea) format. The code to process GPS information is slightly different from that described by reference [7].

The GPS message is received over RS-232 interface via COM1 serial port. The message length of 8-channel GPS data is 76 bytes long. The GPS object instantiates the GPS buffer and the serial port object, which communicates with the GPS receiver. The GPS object checks for the arrival of new messages. Before the GPS object recognizes a message as valid, the message must pass the conditions below:

- The message should have the proper header for the Motorola position message format.

- The message has a valid checksum.

- The number of satellites tracked must be at least three.

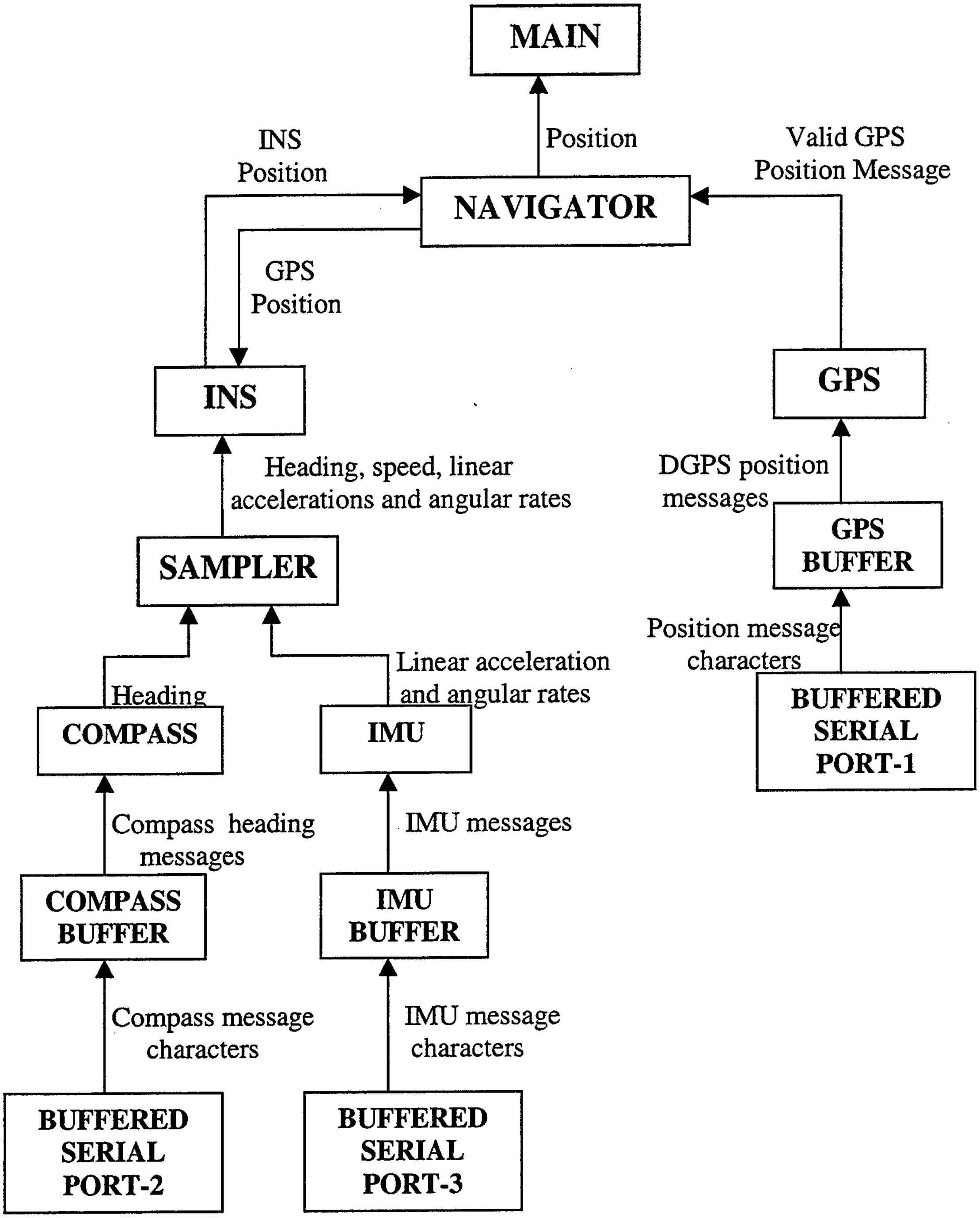

Figure 3.1 SANS Software Objects and Data Flow

- The differential receiver status message bit in GPS message must be set.

If one of these conditions fails, the message is considered invalid.

2. Compass Data

The compass data is received over RS-232 interface via the COM2 serial port. The code to acquire compass data is the same as described in reference [7]. The object instantiates the compass buffer and serial port objects needed to communicate with the compass. The message length for compass data is 60 bytes long. When a compass message is received at the communications port, the code checks the checksum and the header. If one of these checks fails, the message is considered invalid and is ignored.

3. IMU Data

The IMU data is received over RS-232 interface via the COM3 serial port. The IMU object instantiates the IMU buffer and serial port objects needed to communicate with the Crossbow DMU-VG IMU. Each data packet of the IMU begins with a header byte, and ends with a checksum. When an IMU message arrives at the communication port, the code checks the header, and calculates the checksum and compares it to the checksum of the data packet. If one of these checks fails, the message is considered invalid and is ignored.

The Crossbow DMU-VG IMU can operate in one of three modes: voltage mode, scaled sensor mode, or VG mode. The SANS system uses VG mode. The IMU data packet format for VG mode is shown in Table 2.3. The message has 22 bytes of data. The data packet consists of stabilized pitch and roll angles along with angular rate and linear acceleration information.

The digital data is received as a 16-bit number (two bytes). The MSB of the data is received first, followed by the LSB. This digital data can be converted into a single number using the following equation:

$$\text{value} = \text{MSB} \times 256 + \text{LSB} \tag{3.1}$$

The acceleration data (x, y, and z) in data packet is converted into G's (gravity). The digital data is first converted into a single number using equation (3.1). Then, the following equation is used:

$$\text{acceleration} = \text{value} \times (\text{GR} \times 1.5) / 2^{15} \tag{3.2}$$

"GR" is the G range of the IMU unit. It is 2G for the IMU used in the SANS system.

The angular rate data (roll, pitch, and yaw rate) in the data packet is converted into degrees per second. The digital data is first converted into a single number using equation (3.1). Then, the following equation is used:

$$\text{angular rate} = \text{value} \times (\text{AR} \times 1.5)/ 2^{15} \tag{3.3}$$

"AR" is the angular rate range of the IMU unit. It is 50 degrees per second for the IMU used in the SANS system.

The IMU has a simple command structure. A command consisting of one or two bytes can be sent to the sensor over the RS-232 interface. Table 3.1 shows the DMU-VG six-axis IMU command sets.

4. INS

The INS class implements the inertial navigation portion of the SANS using the asynchronous Kalman filter. The INS class instantiates a Sampler object from which it obtains heading, speed, linear acceleration data, and angular rate data. GPS information is also passed to the INS class via Navigator object.

Command (ASCII)	Response	Description
R	H	Reset: Resets the DMU
G	Data Packet	Get Data: Requests a packet of data from The DMU.
r	R	Change to Voltage Mode
c	C	Change to Scaled Sensor Mode
a	A	Change to VG output Mode
T<0-255>	None	2 byte command sequence that changes the vertical gyro erection rate.
C	None	Change to continuous data transmit mode. Data packets streamed continuously.
P	None	Change to polled mode. Data packets sent when G is received by the DMU.
z<0-255>	Z	Calibrate and set zero bias rate sensors by averaging over time. 1st byte initiates zeroing process. 2 nd byte sets duration for averaging.

Table 3.1 IMU Command Sets [Ref. 19]

The INS produces accurate navigation information by integrating IMU data and DGPS data. While IMU data is sampled continuously, DGPS data is available only aperiodically due to asynchronous reacquisition of satellite signals and asynchronous submergence of the AUV. Asynchronous Kalman filtering is an ideal method to obtain accurate navigation information. [Ref. 3]

Figure 3.2 presents a data flow diagram of the SANS navigation filter. The asynchronous Kalman Filter has six states for orientation estimation, and eight states for position estimation. The orientation estimation part of the filter remains the same as described in reference [6] and will not be presented here.

The position estimation part of the filter uses the measurement of the velocity relative to water provided by the water speed sensor and position information provided by DGPS. The velocity measurements are synchronous and available at every sampling interval. DGPS information is asynchronous and is only available when the AUV is surfaced.

The Kalman filter is a recursive predictive update technique used to estimate the states of a process model. Given some initial estimates, it allows the states of a model to be predicted and adjusted with each new measurement.

The filter contains five recursive equations. The Kalman filter gain ($\mathbf{K}_k$) is needed to find the optimal estimated states ($\hat{\mathbf{x}}_k$). It takes the error covariance (mean-square error) between the current state $\mathbf{x}_k$ and the estimated state $\hat{\mathbf{x}}_k$ and applies it to the $\mathbf{H}$ and $\mathbf{R}$ matrixes resulting in

$$\mathbf{K}_k = \mathbf{P}_k^{-}\mathbf{H}^T(\mathbf{H}\ \mathbf{P}_k^{-}\mathbf{H}^T + \mathbf{R}\ )^{-1} \qquad (3.4)$$

Beginning with a prior estimate $\hat{\mathbf{x}}^{-}{}_k$, the noisy measurement $\mathbf{z}_k$ is used with a blending factor $\mathbf{K}_k$ to improve the estimate as follows;

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^{-} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}\ \hat{\mathbf{x}}_k^{-}) \qquad (3.5)$$

The vector $\mathbf{x}_k$ consists of eight states which are the north position, east position, north velocity, east velocity, north current, east current, north GPS bias, and east GPS bias.

Once the Kalman gain $\mathbf{K}_k$ minimizes the mean-square estimation error, the error covariance matrix for $\hat{\mathbf{x}}_k$ can be computed using the equation below;

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}\ )\mathbf{P}_k^{-} \qquad (3.6)$$

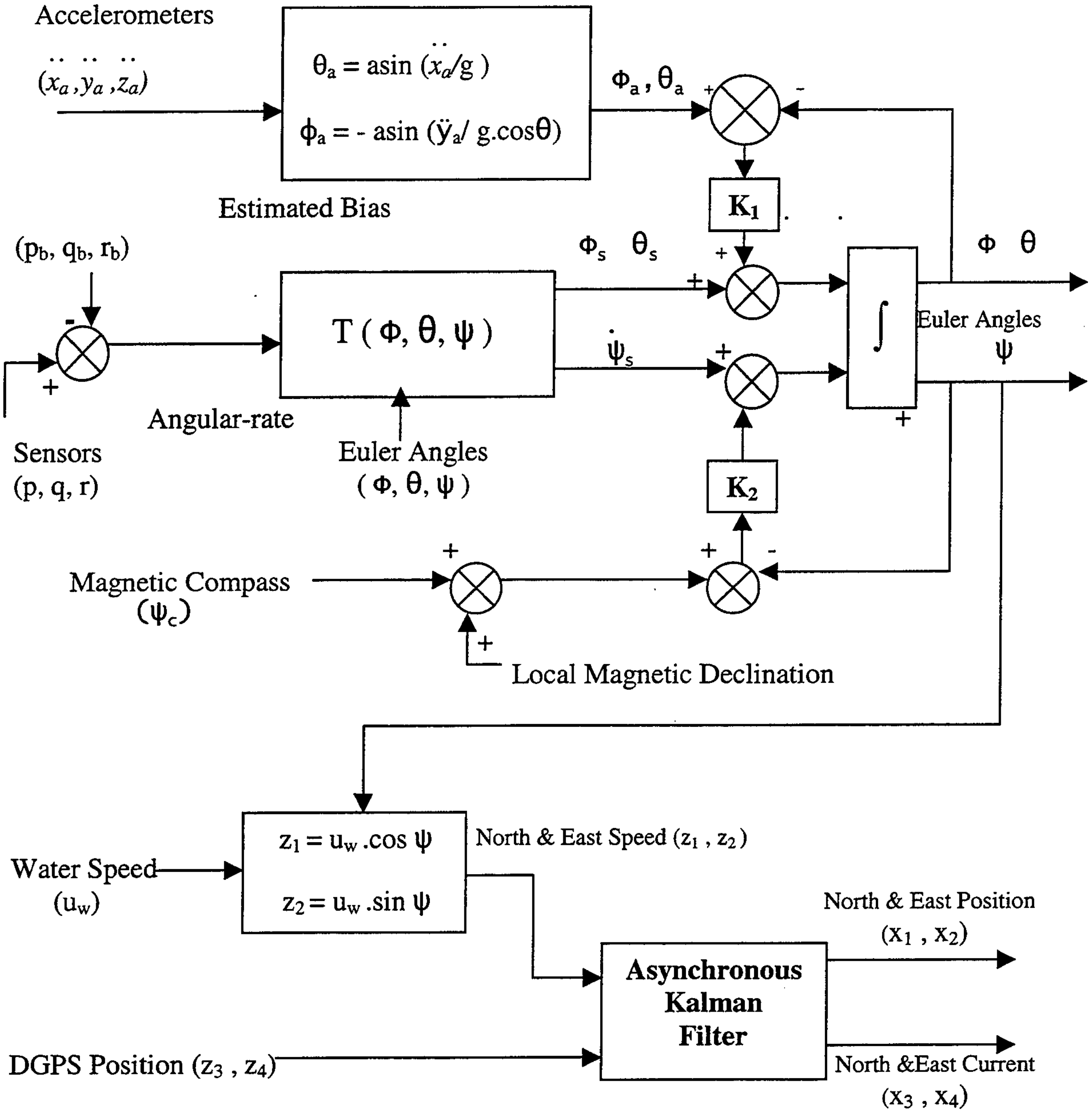

Figure 3.2 SANS Navigation Filter

The updated estimate $\hat{\mathbf{x}}_k$ is projected ahead using the state transition matrix ϕ.

Thus,

$$\hat{\mathbf{x}}_{k+1}^{-} = \phi\ \hat{\mathbf{x}}_k \tag{3.7}$$

Finally, the projected error covariance for $\hat{\mathbf{x}}_{k+1}^{-}$ can be calculated as follows:

$$\mathbf{P}_{k+1}^{-} = \phi_k \mathbf{P}_k \phi_k^T + \mathbf{Q}_k \tag{3.8}$$

Equations (3.4 – 3.8) are used as an algorithm that loops infinitely. This Kalman filter loop is shown in Figure (3.2).

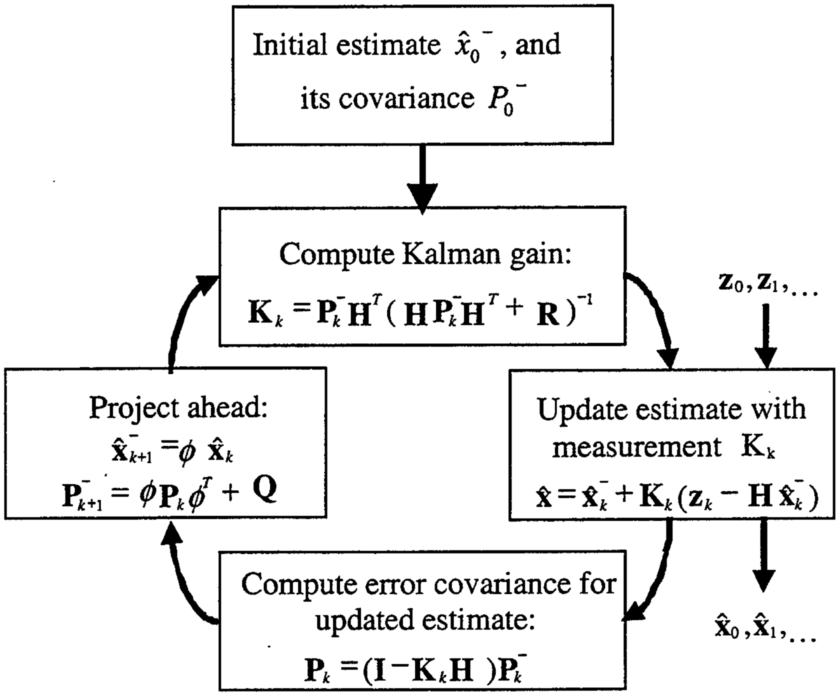
Initial estimate $\hat{x}_0^{-}$, and its covariance P_0^{-}
Compute Kalman gain:
$\mathbf{K}_k = \mathbf{P}_k^{-}\mathbf{H}^T(\mathbf{H}\mathbf{P}_k^{-}\mathbf{H}^T + \mathbf{R})^{-1}$
$\mathbf{z}_0, \mathbf{z}_1, \ldots$
Update estimate with measurement $\mathbf{K}_k$
$\hat{\mathbf{x}} = \hat{\mathbf{x}}_k^{-} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^{-})$
$\hat{\mathbf{x}}_0, \hat{\mathbf{x}}_1, \ldots$
Compute error covariance for updated estimate:
$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^{-}$
Project ahead:
$\hat{\mathbf{x}}_{k+1}^{-} = \phi\ \hat{\mathbf{x}}_k$
$\mathbf{P}_{k+1}^{-} = \phi \mathbf{P}_k \phi^T + \mathbf{Q}$

Figure 3.3 Kalman Filter Loop [Ref. 3]

The state transition matrix ϕ can be calculated as follows;

$$\phi = \begin{bmatrix} e^{\frac{-|\Delta t|}{\tau_1}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & e^{\frac{-|\Delta t|}{\tau_1}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & e^{\frac{-|\Delta t|}{\tau_2}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{\frac{-|\Delta t|}{\tau_2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & e^{\frac{-|\Delta t|}{\tau_3}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{\frac{-|\Delta t|}{\tau_3}} & 0 & 0 \\ \tau_1\left(1-e^{\frac{-|\Delta t|}{\tau_1}}\right) & 0 & \tau_2\left(1-e^{\frac{-|\Delta t|}{\tau_2}}\right) & 0 & 0 & 0 & 0 & 0 \\ 0 & \tau_1\left(1-e^{\frac{-|\Delta t|}{\tau_1}}\right) & 0 & \tau_2\left(1-e^{\frac{-|\Delta t|}{\tau_2}}\right) & 0 & 0 & 0 & 0 \end{bmatrix}$$

The next item needed for the Kalman filter was the $\mathbf{Q}_k$ matrix.

$$\mathbf{Q}_k = \begin{bmatrix} \frac{1}{2\tau_1}\left(1-e^{-\frac{2|\Delta t|}{\tau_1}}\right) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2\tau_1}\left(1-e^{-\frac{2|\Delta t|}{\tau_1}}\right) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2\tau_2}\left(1-e^{-\frac{2|\Delta t|}{\tau_2}}\right) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2\tau_2}\left(1-e^{-\frac{2|\Delta t|}{\tau_2}}\right) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2\tau_3}\left(1-e^{-\frac{2|\Delta t|}{\tau_3}}\right) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2\tau_3}\left(1-e^{-\frac{2|\Delta t|}{\tau_3}}\right) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The measurement error covariance **R** is estimated as;,

$$\mathbf{R} = \begin{bmatrix} .5 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{with DGPS signal}$$

or

$$\mathbf{R} = \begin{bmatrix} .5 & 0 \\ 0 & .5 \end{bmatrix} \text{without DGPS signal}$$

The matrix **H** is the ideal (noiseless) connection between the measurements and the state vector at time *t*. Two **H** matrices describe this connection, one for samples with DGPS the other for samples without DGPS.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \text{with DGPS signal}$$

or

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \text{without DGPS signal}$$

5. Sampler

The Sampler prepares raw IMU, heading and water speed data for use by the INS object. The Sampler controls the data formatting and returns a formatted sample if valid raw IMU data is available. Otherwise, a negative response is returned.

The sampler instantiates the IMU object from which it obtains IMU data packets as shown in Table 2.3. The data received from the IMU is signed 16 bit 2's complement integers. The sampler object first uses equation (3.1) to obtain single numbers. If the number is greater than 32767 (2^{15}-1), it subtracts 65536 (2^{16}) from the number to get the correct sign. After this operation, the sampler object formats the linear acceleration and angular rate data using equations (3.2) and (3.3).

The sampler object also calculates the time delta (Δt) using the IMU clock. The IMU time data is presented in the 19th and 20th bytes of the IMU data packet. The IMU clock counts down from 65535 to 0. A single tick corresponds to 0.79 microseconds [Ref. 15]. The sampler uses the following equation to calculate Δt:

$$\Delta t = 0.79\times10^{-6} \times \text{time difference} \qquad (3.9)$$

where "time difference" is calculated by subtracting the current IMU time from the previous one.

The sampler object also instantiates the compass object from which it obtains heading information. The compass object is unchanged from that described in Reference [7].

6. Navigator

The navigator object instantiates both GPS and INS objects and provides an estimate of the current position in hours, minutes, seconds and milliseconds of latitude and longitude. The navigator object is invoked by the main object. The navigator object interfaces with the GPS and INS objects to determine if they have an updated estimate of the current position. If GPS information available, the navigator object converts a latitude and longitude expressed in milliseconds to a grid position in feet and passes it to INS class, so that the INS object can calculate the current position estimate with new GPS information. If no GPS information is available, the INS object calculates the current position estimate without GPS fixes. If the navigator object obtains an updated grid position from the INS, it converts the information to degrees, minutes, seconds and milliseconds and returns this as the current estimated position. If no updated INS information is available, the navigator returns a negative reply indicating that there is no updated position estimate.

C. SEALEVEL C4-104 I/O MODULE SOFTWARE

The C4-104 module provides four RS-232 serial ports, utilizing a 16554 UART. A UART contains seven functional registers that are used for reporting the ports' status and for initializing the communication parameters under which the serial port will function. In order to access to these registers, the DOS operating system reserves locations in memory which hold the base address for the UART associated with COM1-COM4.

The SETCOM program is a special program provided by manufacturer for the C4-104 module. It can be used to set COM1-COM4 address and communication parameters. This program initializes the COM port addresses. It also sets the baud rate and other parameters needed to communicate. The baud rate can be set up to values as high as 115.2 K baud.

The syntax of the SETCOM program is given below;

```
SETCOM A, BBB, CCCC, D, E, F
```

where, A is the COM port number to be set (1-4), B is the Hex Address of the COM port, C is the baud rate (300,600,1200,4800,9600,19.2,38.4,57.6,115.2), D is the parity (N for no parity, E for even parity, and O for odd parity), E is the word length in bits (5,6,7,8), and F is stop bits (1,2).

As an example to set COM1 to address 2F8 Hex, 19.2 K baud rate, no parity, 7 bit word, and 1 stop bit, the SETCOM command would be;

```
SETCOM  1, 2F8, 19.2, N, 7, 1
```

When the PC is first started, the communication ports are initialized using SETCOM program. The SETCOM program settings for the SANS are shown in Table 3.2. The SETCOM program is put in the “autoexec.bat” file of computer, so that each time the PC is started, it can set the ports automatically.

Port No	Adress (hex)	Baud Rate	Parity	Word Length (bit)	Stop Bits
1	3F8	9600	No Parity	8	1
2	2F8	9600	No Parity	8	1
3	3E8	38400	No Parity	8	1
4	3E8	9600	No Parity	8	1

Table 3.2 SETCOM Program Settings for Serial Port

D. SUMMARY

All additions and updates to the SANS software were compiled under the Borland version 5.0, C^{++} compiler. The software runs on a DOS (standard) platform with an AMD 586DX/133 MHz processor.

Significant as well as minor changes have been made in the SANS software. The IMU data is now received from a serial port. Replacing the previous unit with the new IMU eliminated the need for additional code to operate the A/D converter module, and buffers this data. The compass data and GPS data are also received via serial ports. The code to acquire water speed sensor data is currently under development.

Since DGPS information is available aperiodically due to asynchronous reacquisition time of satellite signals and asynchronous submergence and surfacing duration of the AUV, an asynchronous Kalman filter is needed to optimally integrate IMU and DGPS data. The previous SANS constant-gain filter is now replaced by an asynchronous Kalman filter. This filter has six states for orientation estimation (still constant gain), and eight states for position estimation.

A complete copy of all SANS software is presented in Appendix A and B.

IV. SYSTEM TESTING

A. INTRODUCTION

This chapter presents the bench testing of the current SANS configuration. After integrating the new hardware and implementing the new software, bench testing was performed to determine the functionality and accuracy of the entire system. Simulation results of the SANS Kalman filter are presented in the following section. The system was tested with different speed and different heading information.

B. BENCH TESTING

The system was tested with north heading and east heading information, and position versus time plotting was presented. For bench testing, water speed sensor was simulated by applying different voltages to the system for different speed information. The speedometer developed in reference [6] was utilized in previous versions of the SANS to measure the speed of the system. In order to simulate a speedometer for the bench testing, the following equation was used to convert voltage (V) into speed (v):

$$v = \frac{-7.64}{V} \tag{4.1}$$

The voltage value applied to the system is transferred into a DM406 A/D converter PC/104 module. The DM406 A/D converter module converts the analog voltage input into 12-bit twos complement form. Then, the result must be converted to straight binary. The conversion from twos complement form to straight binary formula is simple: for values greater than 2047, 4096 is subtracted from the value to get the sign of the voltage. Each bit of the A/D module represents 2.44 millivolts. Therefore, the signed voltage is multiplied by 0.00244 to obtain input voltage values. Finally, equation (4.1) is calculate speed values. All these calculations are performed in the sampler object of the SANS software.

Figure 4.1 shows the estimated position against time. The system is tested with the north heading and three feet per second speed input. The figure shows that the north position is increasing by almost three feet per second, and the east position is almost zero.

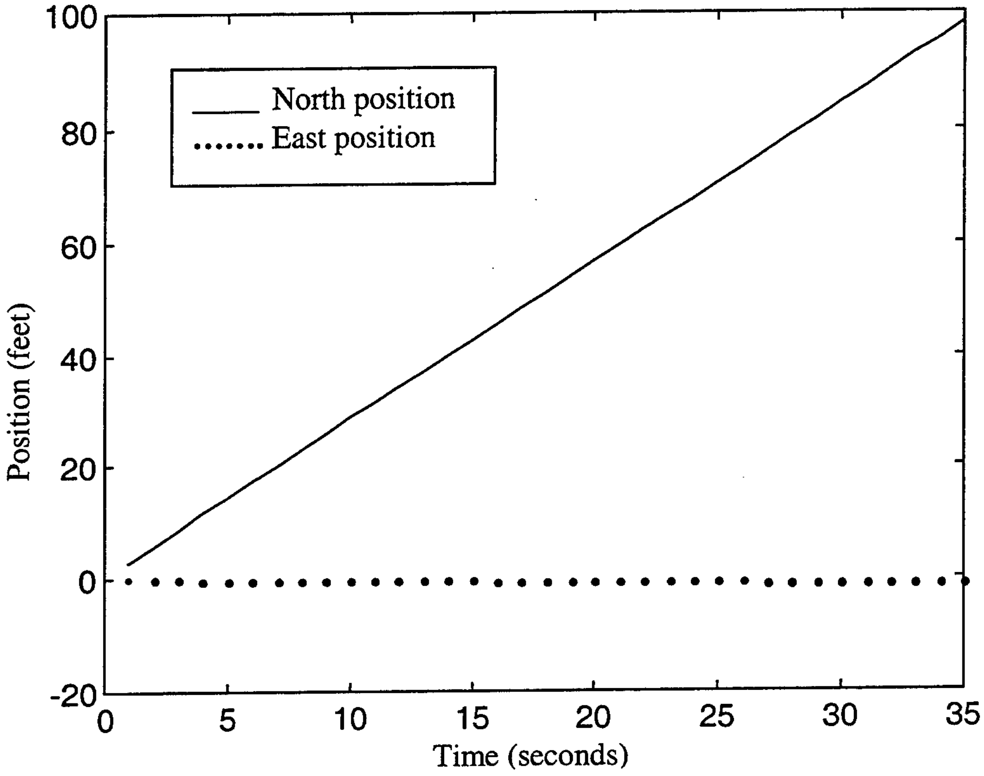

Figure 4.1 Plot of Position vs. Time with the Speed of 3 ft/sec.

Figure 4.2 shows the estimated position against time. The system is tested with the north heading and ten feet per second speed input. The figure displays that the north position is increasing by almost ten feet per second, and the east position is almost zero.

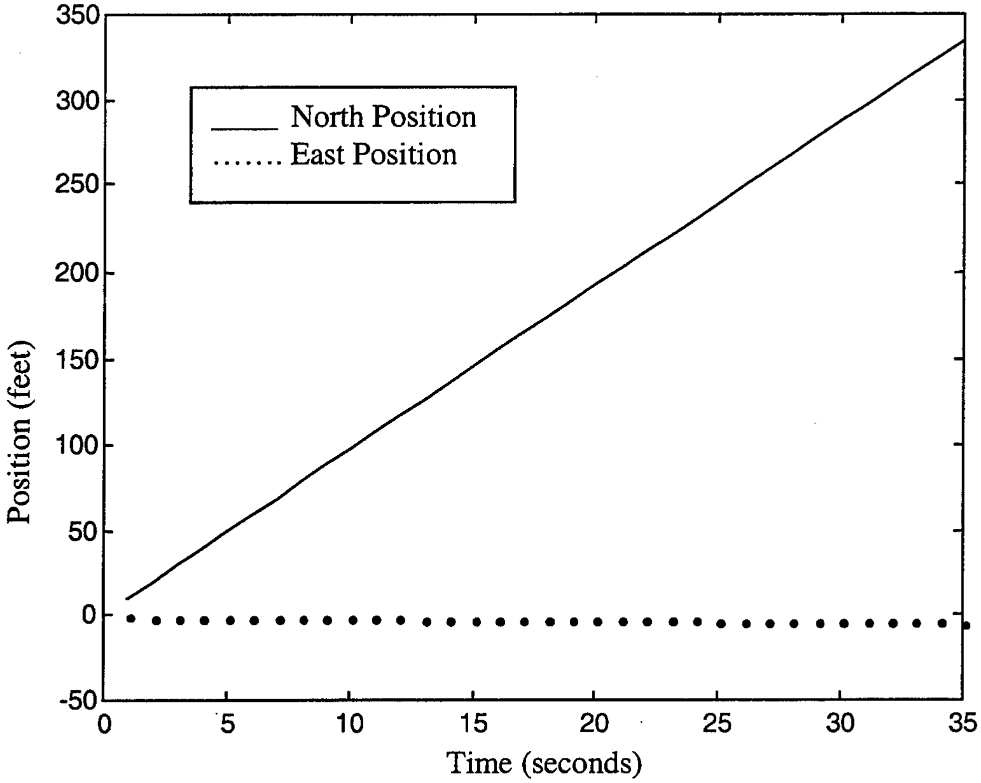

Figure 4.2 Plot of Position vs. Time with the Speed of 10 ft/sec.

Similarlly, the system was tested with east heading information. Figure 4.3 shows the estimated position against time for three feet per second, and figure 4.4 shows the estimated position against time for ten feet per second. Both tests indicate that the east position increases with respect to speed information, and north position is almost zero.

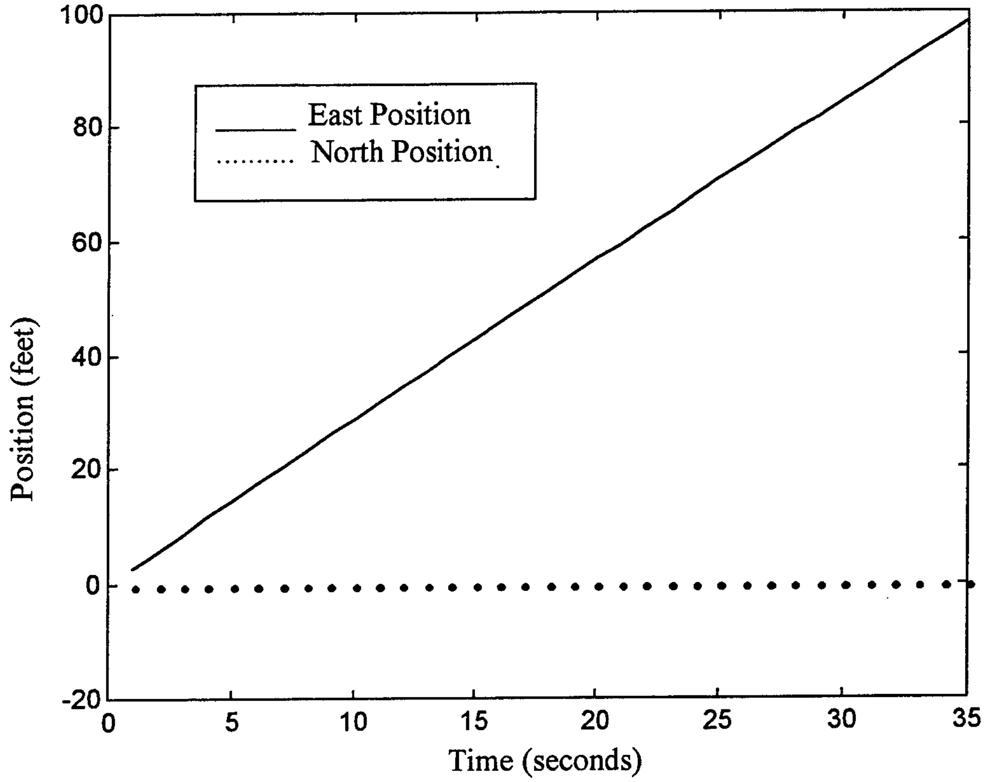

Figure 4.3 Plot of Position vs. Time with the Speed of 3 ft/sec.

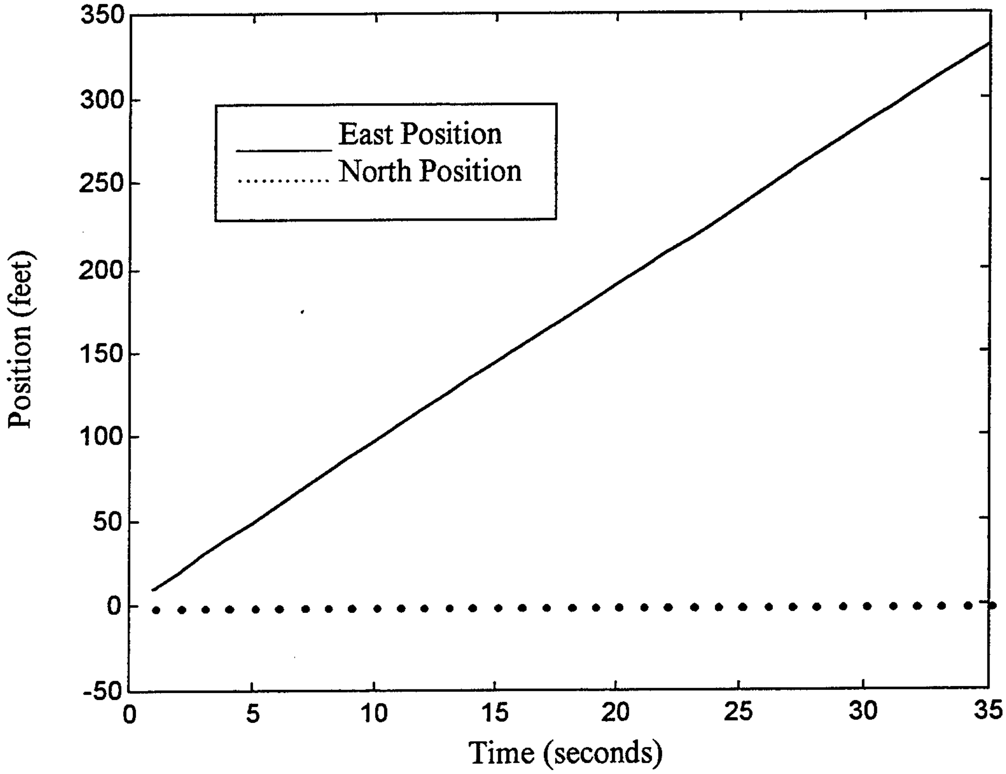

Figure 4.4 Plot of Position vs. Time with the Speed of 10 ft/sec.

C. SUMMARY

The new SANS configuration was tested on a bench with north and east heading information. The bench testing proved that the INS object properly calculates the estimated position using asynchronous Kalman filter.

V. CONCLUSIONS

A. SUMMARY

The purpose of this thesis was to develop a prototype hardware platform and software interface designed to meet the mission requirements of the SANS. The objective of designing the SANS system is to demonstrate the feasibility of using low-cost, small components to navigate inertially between DGPS fixes.

The research issues addressed by this thesis were: (1) Integrate an AMD 586DX133 based PC/104 computer into the SANS system, (2) Develop the software interface, which communicates between the GPS receiver, the compass, the IMU and, the processing computer, (3) Implement the asynchronous Kalman filter developed in reference [3] into the new hardware system, and (4) Test and evaluate the hardware and software.

The work conducted in addressing the first of these research issues resulted in a new hardware configuration of the SANS system. The new hardware configuration uses an AMD 586DX133 based PC/104 computer, a C4-104 Serial I/O Module and a Crossbow DMU-VG Six Axis IMU. The components in SANS were chosen based on cost, size, and ease of operation. The new components reduced the size of the system by 52% and increased the sampling rate to more than 80Hz. Use of the PC/104 industrial standard enhanced the reliability, flexibility, and compatibility of the SANS system.

In addressing the second research issue, some significant as well as minor changes were made in the SANS software. In the current system, the IMU data, compass data and GPS data are all received via serial port communication objects. The SANS software was compiled under the Borland version 5.0, C^{++} compiler. The software runs on a DOS (standard) platform.

For the third research question, the previous SANS constant-gain filter was replaced by an asynchronous Kalman filter, which has six states for orientation estimation (still constant gain), and eight states for position estimation. The asynchronous nature of DGPS measurements due to asynchronous reacquisition time of satellite signals and asynchronous submergence and surfacing of the AUV, made the selection of an asynchronous Kalman filter algorithm a logical choice.

After integrating the new hardware and implementing the new software, bench testing was conducted and indicated that the newly designed system provides a higher level of performance than previous versions of SANS. The examination of the experimental data indicates that the new IMU used in this research is capable of meeting all SANS requirements. The new data acquisition and processing unit increased the speed, reliability, and compatibility of the system. Testing the new asynchronous Kalman filter with different speed and heading data indicates that the new navigation filter works properly.

B. FUTURE RESEARCH

Technology advances, software development, and the amount of research put into testing and evaluation show that the future of SANS is subject to many changes. The goal of choosing the new components in the SANS system should always be to reduce the size, while improving performance and decreasing cost.

Addition of the Proxim Range LAN2 PC card to the system should be considered as a future component. This card would give SANS more portability and potentially change the way the sensors are integrated and utilized for applications other than AUVs.

In order to acquire more accurate navigation information, the four year old DGPS package should be replaced with a system that is smaller and more accurate.

The asynchronous Kalman filter has been implemented as a navigation filter. More testing of the filter is needed, in order to adjust filter constants.

The new water speed sensor must be integrated into the system. The water speed sensor should also use a serial port communication object to obtain water speed information.

Tilt table tests must be performed to examine the IMU data. Compass calibration is examined in reference [6]. When SANS reaches the stage where hardware and software are fully integrated, at-sea trials will be needed to prove its operation.

APPENDIX A: REAL TIME NAVIGATION SOURCE CODE(C++)

A. TOETYPES.H

```
#ifndef __TOETYPES_H
#define __TOETYPES_H

#include "globals.h"    //Types used by serial communications software
#define GPSBLOCKSIZE 76 //Size of Motorola @@Ea position  message
#define CRBBLOCKSIZE 22
#define PACKETSIZE 133  // Size of packet received via X-modem protocol
#define COMPSIZE 60
#define ONE_G 32.2185   // One g in feet per second per sec.
#define GRAVITY 32.2185     // In feet per second per sec.
#define TicksToSecs(x) ((double) ((10 * x) / 182))

typedef char  ONEBYTE;
typedef short TWOBYTE;
typedef long  FOURBYTE;
typedef unsigned char  UNSIGNED_ONEBYTE;
typedef unsigned short UNSIGNED_TWOBYTE;
typedef unsigned long  UNSIGNED_FOURBYTE;

// Holds lat/long expressed in miliseconds
struct latLongMilSec {
    long latitude;
    long longitude;
};

// Holds a latitude or longitude expressed in hours minutes and degrees
struct T_GEODETIC {
    TWOBYTE          degrees;
    UNSIGNED_TWOBYTE  minutes;
    double            seconds;
};

// Holds a latitude and longitude expressed as T_GEODETICs
struct latLongPosition {
    T_GEODETIC latitude;
    T_GEODETIC longitude;
};

// Holds a grid position
struct grid {
    double x,y,z;
};

// 3 X 3 matrix
struct matrix {
    float element[3][3];
};

// 3 X 1 matrix or vector
struct vector {
    float element[3];
};
```

```
// Oversize area to hold a GPS message
typedef BYTE GPSdata[2 * GPSBLOCKSIZE];

// Oversize area to hold a Crossbow IMU message
typedef BYTE CRBdata[2 * CRBBLOCKSIZE];

// Defines a type for holding compass messages
typedef BYTE compData[2 * COMPSIZE];

//stampedSample structure
struct stampedSample {
    Boolean gpsFlag;          // True if GPS fix obtained
    Boolean insFlag;          // True if INS fix obtained
    latLongPosition navLatLong; //posit in hours,mins,secs
    grid est;                 // position as estimated by the INS
    GPSdata satPosition;                 // the latest GPS position
    CRBdata crossbowData;   // the Crossbow data
    float rawSample[8];     //Original readings for post process
    double sample[11];      // sampler converted sample
    double deltaT;          // delta of the sample
    float bias[3];          // bias corrections
    float current[3];       // error correction current
    float rawVelocity[3];

};
#endif
```

B. TOEFISH.CPP

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <dos.h>
#include <time.h>

#include "toetypes.h"
#include "nav.h"
#include "compport.h"
#include "crbbuff.h"
#include "crb.h"

crbBufferClass buf;
extern compassPortClass port2; // so breakhandler can call destructors
extern gpsPortClass port1;     // clean up on program exit
extern crbPortClass port3;

int  breakHandler(void);
void screenSetUp(void);
void printPosition (const latLongPosition&);
void positOut(stampedSample& posit);

// Write an INS packet and its timeStamp to the outPut file
void writeData(const stampedSample& drPosition, ofstream&, float
elapsedTime);
// Write a GPS message to the outPut file.
void writeGpsData(const GPSdata& satPosition);

// Write data in list format for lisp program
void writeLispData(float deltaT,stampedSample& current,
                      ofstream& lispData);

/*******************************************************************
   PROGRAM:Main
   AUTHOR: Eric Bachmann,Dave Gay,Rick Roberts, Kadir Akyol
   DATE:   11 July 1995, last modified March 1999
   FUNCTION: Drives the navigator and its associated   software.
    Counts the positions & displays each to the screen. Exited only when
    control break (Ctrl c) is entered at the keyboard.
   RETURNS:  0
   CALLED BY: none
   CALLS:initializeNavigator (nav.h)
          navPosit (nav.h)
          printPosition
          breakHandler
*********************************************************************/

int
main ()
{
   crbPortClass read;
   ctrlbrk(breakHandler); // trap all breaks to release com ports
   setcbrk(1);            // turn break checking on at all times
```

```
char dataFile[] = "att.dat";
char lispFile[] = "lisp.dat";

cout <<"\nWriting attitude data to " << dataFile <<endl;

// Instantiate the navigator
navigatorClass *navPtr = new navigatorClass;
navigatorClass &nav1 = *navPtr;

ofstream attitudeData(dataFile);
ofstream lispData(lispFile);

stampedSample curLoc;  // Lat/Long of most recent fix

curLoc.navLatLong.latitude.degrees  = 0.0;
curLoc.navLatLong.latitude.minutes  = 0.0;
curLoc.navLatLong.latitude.seconds  = 0.0;
curLoc.navLatLong.longitude.degrees = 0.0;
curLoc.navLatLong.longitude.minutes = 0.0;
curLoc.navLatLong.longitude.seconds = 0.0;

read.Send(’R’);                  // reset command to IMU
cerr << "Reset is sent to IMU " << endl;

read.Send(’a’);                  // VG mode command to IMU
cerr << "VG mode is selected " << endl;

read.Send(0x7A);                 // ‘z’ command to calibrate and set
cerr << "zeroing..." << endl;    // for rate sensor

read.Send(0xFF);   // 2nd byte of ‘z’ command
cerr << "zeroing..." << endl;
delay(10);

read.Send(0x54);             // ‘T’ command for time constant
cerr << "time constant setting..." << endl;

read.Send(0xFF);             // 2nd byte of ‘T’ command
cerr << "time constant setting..." << endl;
delay(10);

read.Send('C');              // Continuous transmission mode
cerr << "Continuous mode " << endl;

Boolean  gpsReceived(FALSE);     // True if gps received
Boolean  fixReceived(FALSE);     // True if a new fix was received

int      fixCount(0);             // Count of navigation fixes
float    timeCount(0.0);          // Counter for screen output
float    timeTotal(0.0);          // Total elapsed time

cerr << "\nInitializing . . ." << endl;

nav1.initializeNavigator(curLoc);

clrscr();
```

```
   gotoxy(1,6);
   cerr << "Initialization Complete!" << endl;
   cout << "Initial Position:" << endl;

   // Print the initial position
    cout << "latitude: " <<
    curLoc.navLatLong.latitude.degrees << ':'
       << curLoc.navLatLong.latitude.minutes << ':'
       << curLoc.navLatLong.latitude.seconds << endl;
    cout << "longitude: " << curLoc.navLatLong.longitude.degrees << ':'
       << curLoc.navLatLong.longitude.minutes << ':'
       << curLoc.navLatLong.longitude.seconds;

   screenSetUp();

   //Attempt to get a fix from the navigator
   while (TRUE) {
   fixReceived = nav1.navPosit(curLoc);

      if (fixReceived) {         // New fix received
      // Print info to screen at designated print interval
           fixCount++;
           timeCount += curLoc.deltaT;
           timeTotal += curLoc.deltaT;

           if (curLoc.gpsFlag) {
              gpsReceived = TRUE; //Keep DGPS indicator displayed
              writeLispData(timeCount, curLoc, lispData);
           }

           if (timeCount >= 1.0) {

              if (gpsReceived == TRUE) {
                 gotoxy(20,11);
                 cout << "DGPS";
                 gpsReceived = FALSE;
              }
              else {
                 gotoxy(20,11);
                 cout << "    ";
              }

              gotoxy(9,11);
              cout << fixCount << endl;

              positOut(curLoc);
              writeData(curLoc, attitudeData, timeTotal);
              writeLispData(timeCount, curLoc, lispData);
              timeCount = 0.0;

           } // end if
       } // end if
   } //end while
}// end main
```

```
/***************************************************************
   PROGRAM:   printPosition
   AUTHOR:    Eric Bachmann, Dave Gay
   DATE:      11 July 1995
   FUNCTION:  Displays position to the screen
   RETURNS:   void
   CALLED BY: main
   CALLS:     none
***************************************************************/

void printPosition (const latLongPosition& posit)
{
   gotoxy(11,14);
   cout << posit.latitude.degrees << ':'<<
          posit.latitude.minutes << ':' << posit.latitude.seconds <<
          endl;
   gotoxy(12,15);
   cout << posit.longitude.degrees << ':'<<
         posit.longitude.minutes << ':' << posit.longitude.seconds <<
         endl;
}

/**************************************************************
   PROGRAM:   breakHandler
   AUTHOR:Eric Bachmann,Dave Gay, Rick Roberts, Kadir Akyol
   DATE:         11 July 1995 last modified March 1999
   FUNCTION:  Cleans up com ports upon program exit.
   RETURNS:   0
   CALLED BY: main
   CALLS:     compass port and gps port destructors
**************************************************************/

int breakHandler(void)
{
   crbPortClass re;
   re.Send('R');          // Send reset command to IMU
   delay(100);
   buf.~crbBufferClass();  // clears the crossbow buffer
   port3.~crbPortClass();
   port2.~compassPortClass();
   port1.~gpsPortClass();
   return 0;              // keep the compiler happy
}

/**************************************************************
    PROGRAM:  screenSetup
    AUTHOR:   Eric Bachmann, Randy Walker
    DATE:     12 May 1996
    FUNCTION: Sets up the output screen
    RETURNS:  0
    CALLED BY:main
    CALLS:    none
**************************************************************/

void screenSetUp(void)
{
   gotoxy(4,11);
   cout << "Fix ";

   gotoxy(1,14);
   cout << "Latitude: " << "\nLongitude: ";

   gotoxy(1,17);
   cout << "Roll: " << "\nPitch: ";

   gotoxy(1,25);
   cout << "deltaT: ";

   int col(45),row(1);

   gotoxy(col,row++);
   cout << "x accel: ";
   gotoxy(col,row++);
   cout << "y accel: ";
   gotoxy(col,row++);
   cout << "z accel: ";
   gotoxy(col,row++);
   cout << "phi dot: ";
   gotoxy(col,row++);
   cout << "theta dot: ";
   gotoxy(col,row++);
   cout << "psi dot: ";
   gotoxy(col,row++);
   cout << "water speed: ";
   gotoxy(col,row++);
   cout << "heading: ";

   col = 45;
   row = 12;

   gotoxy(col,row++);
   cout << "x: ";
   gotoxy(col,row++);
   cout << "y: ";
   gotoxy(col,row++);
   cout << "z: ";
   gotoxy(col,row++);
   cout << "phi: ";
   gotoxy(col,row++);
   cout << "theta: ";
   gotoxy(col,row++);
   cout << "psi: ";

   gotoxy(45,20);
   cout << "Bias Values";

   gotoxy(60,20);
   cout << "Current Values";
}
```

```
/*******************************************************************
   PROGRAM:    positOut
   AUTHOR:     Eric Bachmann
   DATE:       21 October 1996
   FUNCTION:   Updates the Screen
   RETURNS:    0
   CALLED BY:  main
   CALLS:      none
*******************************************************************/

void positOut(stampedSample& posit)
{
   printPosition(posit.navLatLong);
   int j;

   // Output the bias values
   for(j = 3; j < 6; j++) {
      gotoxy(45,j+18);
      cout << posit.bias[j];
   }

   //Display linear accelrations,angular rates,water speed and comp hdg
   for (j = 0; j < 8; j++) {
      gotoxy(59,j+1);
      cout << posit.rawSample[j];
   }

   // Display time delta to the screen.
   gotoxy(9,25);
   cout << posit.deltaT;

   // Display roll and pitch
   gotoxy(8,17);
   cout << (posit.sample[3] * radToDeg);
   gotoxy(8,18);
   cout << (posit.sample[4] * radToDeg);
   // Display current location and posture
   for (j = 0; j < 6; j++) {
      gotoxy(52,j+12);
      cout << posit.sample[j];
   }
   // Display error current values
   for (j = 0; j < 3; j++) {
      gotoxy(60,j+21);
      cout << posit.current[j];
   }

   // Output the biases
   for (j = 0; j < 3; j++) {
      gotoxy(45,j+21);
      cout << posit.bias[j];
   }
}
```

```
/*********************************************************************
   PROGRAM:    writeData
   AUTHOR:     Eric Bachmann, Dave Gay
   DATE:       11 July 1995
   FUNCTION:   Writes the packet and the time stamp contained in a
    stamped sample to the out put file for post processing.
   RETURNS:    void
   CALLED BY:  navPosit (nav.cpp)
   CALLS:      None
*********************************************************************/

void writeData(const stampedSample& drPosition,
           ofstream& attitudeData,float elapsedTime)
{
   // Output attitude data to a file
   attitudeData
       << elapsedTime << ' '
       << drPosition.sample[0] << ' '
       << drPosition.sample[1] << ' '
       << drPosition.sample[2] << ' '
       << (radToDeg * drPosition.sample[3]) << ' '
       << (radToDeg * drPosition.sample[4]) << ' '
       << (radToDeg * drPosition.sample[5]) << ' '
       << drPosition.sample[6] << ' '
       << (radToDeg * drPosition.sample[7]) << ' '
       << drPosition.current[0] << ' '
       << drPosition.current[1] <<endl;
}

/*******************************************************************
   PROGRAM:    writeData
   AUTHOR:     Eric Bachmann, Dave Gay
   DATE:       11 July 1995
   FUNCTION:    Writes the packet and the time stamp contained in a
    stamped sample to the out put file for post processing.
   RETURNS:     void
   CALLED BY:  navPosit (nav.cpp)
   CALLS:      None
*******************************************************************/

void writeLispData(float deltaT,stampedSample& current,
          ofstream& lispData)
{
   // Output attitude data to a file
   if (current.gpsFlag) {
      lispData
       << '(' << deltaT << ' '
       << current.rawVelocity[0] << ' '
       << current.rawVelocity[1] << ' '
       << current.est.x << ' '
       << current.est.y << ')'  << endl;
   }
   else {
      lispData
       << '(' << current.deltaT << ' '
       << current.rawVelocity[0] << ' '
       << current.rawVelocity[1]
```

```
            << " nil nil)"<< ' '  << endl;
    }
}
// end of file toefish.cpp
```

C. NAV.H

```
#ifndef _NAVIGATOR_H
#define _NAVIGATOR_H
#include <stdio.h>
#include <fstream.h>
#include <iostream.h>
#include <math.h>
#include <dos.h>

#include "toetypes.h"
#include "globals.h"
#include "gps.h"
#include "ins.h"

/*******************************************************************
   CLASS:      navigatorClass
   AUTHOR:     Eric Bachmann, Dave Gay, Rick Roberts
   DATE:       11 July 1995, Modified January 1997
   FUNCTION:   Combines GPS and INS information to return the current
               estimated position.
*******************************************************************/
class navigatorClass {

   public:

      // Constructor, initializes object slots
      navigatorClass() : gpsSpeedSum(0.0), insSpeedSum(0.0)
       { cerr << "\nconstructing nav1" << endl; };

      ~navigatorClass() {}                  // Destructor

      // provides the navigator's best estimate of current position
      Boolean navPosit (stampedSample&);

      // Initialize the navigator
      Boolean initializeNavigator(stampedSample&);

      void userInitNav(stampedSample&); //Allows user to initialize nav

   private:

      double gpsSpeed, insSpeed, gpsSpeedSum, insSpeedSum;

      insClass ins1;          // ins object instance
      gpsClass gps1;          // gps object instance

      // Obtains system time to utilize for origin
      double getSystemTime();

      latLongMilSec origin; //lat-long of navigational origin

      // Returns the position in Miliseconds
      latLongMilSec getMilSec(const GPSdata&);


      // Returns the position in degrees, minutes, seconds and milisecs
      latLongMilSec latLongToMilSec(const latLongPosition&);
      // Convert position in milSec to degress, minutes, seconds and
milsec
      latLongPosition milSecToLatLong(const latLongMilSec&);

      // Convert xy (grid) position to lat long
      latLongMilSec gridToMilSec(const grid&);

      // Converts lat/long to xy position
      grid milSecToGrid(const latLongMilSec&);

      // Parses and returns the time of a GPS message.
      double getGpsTime(const GPSdata& rawMessage);

      // Parses and returns the velocity of a GPS message.
      double getGpsVelocity(const GPSdata& rawMessage);
};
#endif
```

D. NAV.CPP

```
#include <signal.h>
#include <dos.h>
#include <time.h>
#include <stdlib.h>
#include "nav.h"

#define SIGFPE 8             // Floating point exception

/*******************************************************************
   PROGRAM:  navPosit
   AUTHOR:   Eric Bachmann, Dave Gay, Kadir Akyol
   DATE:     11 July 1995 last modified March 1999
   FUNCTION: Provides the navigator's best estimate of current
   position. Attempts to obtain GPS and INS position fixes from the gps
   and ins objects and copies the most accurate fix available into the
   input argument 'navPosition'.  Sets a return flag to indicate
   whether a valid fix was obtained.
   RETURNS:   TRUE, a valid position fix is in the variable
   'navPosition'. FALSE, otherwise.
   CALLED BY:  towfish.cpp (main)
   CALLS:      gpsPosition (gps.h)        gridToMilSec (nav.h)
               correctPosition (ins.h)    milSecToGrid (nav.h)
               insPosition (ins.h)     milSecToLatLong (nav.h)
               getMilSec (nav.h)      writeScriptPosit (nav.h)
*******************************************************************/

void fpeNavPosit(int sig)
    {if (sig == SIGFPE) cerr << "floating point error in navPosit\n";}

Boolean navigatorClass::navPosit (stampedSample& posit)
{
   signal (SIGFPE, fpeNavPosit);

   latLongMilSec gpsMilSec; //latest GPS position in milsec
   latLongMilSec insMilSec; //latest INS position in milsec

   // Attempt to get the INS and GPS positions
   posit.gpsFlag = gps1.gpsPosition(posit.satPosition);

   if (posit.gpsFlag) {       // GPS positions obtained?
      // Parse position from GPS messsage
      gpsMilSec = getMilSec(posit.satPosition);

      posit.est = milSecToGrid(gpsMilSec);

      // Convert position in milisec to latitude and longitude.
      posit.navLatLong = milSecToLatLong(gpsMilSec);

   }
  posit.insFlag = ins1.insPosition(posit);

  if (posit.insFlag) {    // Only INS position obtained?

     insMilSec = gridToMilSec(posit.est);
```

```
      posit.navLatLong = milSecToLatLong(insMilSec);

      insSpeed = posit.sample[6];

      return TRUE;
  }
  else {
      return FALSE;
  }
}

/*****************************************************************
   PROGRAM:    initializeNavigator
   AUTHOR:     Eric Bachmann, Dave Gay, Rick Roberts
   DATE:       11 July 1995
   FUNCTION:   Obtains an initial GPS fix for use as a navigational
    origin for grid positions used by the INS object. Saves the origin
    and passes it to the INS object in latLong form.
   RETURNS:    TRUE
   CALLED BY:  towfish (main)
   CALLS:     gpsPosition (gps.cpp)      writeGpsData(nav.cpp)
              correctPosition (ins.cpp) getMilSec (nav.cpp)
              writeInsData(nav.cpp)      milSecToGrid (nav.cpp)
*****************************************************************/

Boolean navigatorClass::initializeNavigator(stampedSample& posit)
{
   Boolean gpsFlag(FALSE);

   cerr << "Initializing Navigator." << endl;
   cerr << "   Initializing GPS." << endl;

   // Loop until an initial GPS fix is obtained.
   for(int i = 1 ; ((i < 100) &&(gpsFlag == FALSE)); i++) {
      if (gps1.gpsPosition(posit.satPosition)) {
           gpsFlag = TRUE;
      }
      else {
           delay(100);
      }
   }

   if (gpsFlag == FALSE) {
       cerr << "\nWARNING: UNABLE TO OBTAIN INITIAL GPS FIX!" << endl;
       userInitNav(posit);
   }

   cerr << "   GPS initialization complete." << endl;

   // Convert position in milisec to latitude and longitude.
   posit.navLatLong = milSecToLatLong(getMilSec(posit.satPosition));


   // Save navigational origin for later grid position conversions.
   origin = getMilSec(posit.satPosition);
```

```
    // Pass time of first GPS fix to INS object initialization routine.
   ins1.insSetUp(getGpsTime(posit.satPosition), posit);
   cerr << "Navigator initialization complete." << endl;

   return TRUE;
}

/*******************************************************************
   PROGRAM:    userInitNav
   AUTHOR:     Rick Roberts
   DATE:       03 November 1996
   FUNCTION:   Allows user to input current position and initialize nav
    if no gps fix is available.(ie for testing)
   RETURNS:    void
   CALLED BY: initializeNavigator
   CALLS:     getMilSec (nav.cpp), getSystemTime (nav.cpp)
*******************************************************************/

void navigatorClass::userInitNav(stampedSample& posit)
{
   int choice;

    cerr << "\nEnter a 0 to enter posit and continue without GPS"
         << "\nEnter a 1 to continue without GPS or initial posit, or"
         << "\nEnter a 2 to exit: "
         << endl;
   cin >> choice;

   if (choice == 0) {
      cerr << "\nEnter current position in the following format: "
           << endl;
      cerr << "Latitude: (36, Enter, 35 Enter, 41.5 Enter)" << endl;
      cin  >> posit.navLatLong.latitude.degrees;
      cin  >> posit.navLatLong.latitude.minutes;
      cin  >> posit.navLatLong.latitude.seconds;

      cerr << "Longitude: (-121, Enter, 52, Enter, 30.2, Enter)"
           << endl;
      cin  >> posit.navLatLong.longitude.degrees;
      cin  >> posit.navLatLong.longitude.minutes;
      cin  >> posit.navLatLong.longitude.seconds;
   }
   else if (choice == 2) {
      exit(1);
   }

   // Save nav origin for later grid position conversions
   origin = latLongToMilSec(posit.navLatLong);

}
```

```
/**************************************************************
   PROGRAM:    latLongToMilSec
   AUTHOR:     Rick Roberts
   DATE:       22 January 1997
   FUNCTION:  Converts a position expressed in latitude and longitude
   degrees, minutes and seconds to mili seconds & returns it.
   RETURNS:    latLongMilSec
   CALLED BY: userInitNav
   CALLS:      none
*************************************************************/

latLongMilSec navigatorClass::latLongToMilSec(const latLongPosition&
latLong)
{
   latLongMilSec milSec;
   milSec.latitude = (latLong.latitude.degrees *
        DEGREES_TO_MSECS)+(latLong.latitude.minutes *
        MINS_TO_MSECS)+(latLong.latitude.seconds * 1000.0);

   milSec.longitude = (latLong.longitude.degrees *
        DEGREES_TO_MSECS)+(latLong.longitude.minutes *
        MINS_TO_MSECS)+(latLong.longitude.seconds * 1000.0);

   return milSec;
}

/**************************************************************
   PROGRAM:     getSystemTime
   AUTHOR:      Rick Roberts
   DATE:        03 November 1996
   FUNCTION:    Obtains system time to utilize for origin.
   RETURNS:     double (origin time in seconds)
   CALLED BY:   userInitNav
   CALLS:       dos time function
*************************************************************/

double navigatorClass::getSystemTime()
{
   dostime_t* sysTime;      // pointer to dos time structure

   _dos_gettime(sysTime);

   return double((sysTime->hour * 3600.0) + (sysTime->minute * 60.0)+
(sysTime->second));
}

/***********************************************************
   PROGRAM:    getMilSec
   AUTHOR:     Eric Bachmann, Dave Gay
   DATE:       11 July 1995
   FUNCTION:  Extracts a position in miliseconds from a Motorola
   (@@Ba)position contained in the input argument 'rawMessage'.
   RETURNS:    The latitude and longitude in miliseconds.
   CALLED BY:  navPosit (nav.cpp)
               initializeNavigator (nav.cpp)
   CALLS:      none.
**********************************************************/
```

```
latLongMilSec navigatorClass::getMilSec(const GPSdata& rawMessage)
{
   FOURBYTE temps4byte;
   latLongMilSec position;

   temps4byte        = rawMessage[15];
   temps4byte        = (temps4byte<<8) + rawMessage[16];
   temps4byte        = (temps4byte<<8) + rawMessage[17];
   temps4byte        = (temps4byte<<8) + rawMessage[18];

   position.latitude = temps4byte;


   temps4byte        = rawMessage[19];
   temps4byte        = (temps4byte<<8) + rawMessage[20];
   temps4byte        = (temps4byte<<8) + rawMessage[21];
   temps4byte        = (temps4byte<<8) + rawMessage[22];

   position.longitude = temps4byte;

   return position;
}

/****************************************************************
   PROGRAM:   milSecToLatLong
   AUTHOR:    Eric Bachmann, Dave Gay
   DATE:      11 July 1995
    FUNCTION: Converts a position expressed totally in miliseconds to
    degrees, minutes, seconds and miliseconds.
    RETURNS:   The position in degrees, minutes, seconds and
    miliseconds.
   CALLED BY: navPosit (nav.cpp)
   CALLS:     none
****************************************************************/

latLongPosition navigatorClass::milSecToLatLong(const latLongMilSec&
milSec)
{
   latLongPosition  position;

   double degrees, minutes;

   degrees = (double)milSec.latitude * MSECS_TO_DEGREES;
   position.latitude.degrees = (TWOBYTE)degrees;

   if(degrees < 0) {
      degrees = fabs(degrees);
   }
   minutes = (degrees - (TWOBYTE)degrees) * 60.0;
   position.latitude.minutes = (TWOBYTE)minutes;
   position.latitude.seconds = (minutes - (TWOBYTE)minutes) * 60.0;

   degrees = (double)milSec.longitude * MSECS_TO_DEGREES;
   position.longitude.degrees = (TWOBYTE)degrees;
   if(degrees < 0) {
      degrees = fabs(degrees);
   }
   minutes = (degrees - (TWOBYTE)degrees) * 60.0;
   position.longitude.minutes = (TWOBYTE)minutes;
   position.longitude.seconds = (minutes -(TWOBYTE)minutes) * 60.0;

   return position;
}

/*****************************************************************
   PROGRAM:    gridToMilSec
   AUTHOR:     Eric Bachmann, Dave Gay
   DATE:       11 July 1995
    FUNCTION:  Convert a grid position to a latitude and longitude in
    mili-seconds and returns the result.
   RETURNS:    The latitude and longitude in miliseconds.
   CALLED BY:  navPosit (nav.cpp)
   CALLS:      none
*****************************************************************/

void fpeGridToMilSec(int sig)
{if (sig == SIGFPE) cerr << "floating point error in gridToMilSec\n";}

latLongMilSec navigatorClass::gridToMilSec(const grid& posit)
{
   signal(SIGFPE, fpeGridToMilSec);
   latLongMilSec  latLong;

   // converts grid in ft to latitude
   latLong.latitude = origin.latitude + posit.x / LatToFt);
   // converts grid in ft to longitude
   latLong.longitude = origin.longitud + HemisphereConversion *
(posit.y / LongToFt);

   return latLong;
}

/*****************************************************************
   PROGRAM:    milSecToGrid
   AUTHOR:      Eric Bachmann, Dave Gay
   DATE:       11 July 1995
    FUNCTION:  Convert a latitude and longitude expressed in milseconds
    to a grid position in xy coordinates in feet from the origin.
   RETURNS:    The grid position
   CALLED BY:navPosit(nav.cpp),initializeNavigator(nav.cpp)
   CALLS:     none
   COMMENTS:  altitude is always assumed to be zero.
*****************************************************************/

grid navigatorClass::milSecToGrid(const latLongMilSec& posit)
{

   grid  position;

   position.x = (posit.latitude - origin.latitude) * LatToFt;
   position.y = (posit.longitude - origin.longitude) * LongToFt;
   position.z = 0;

   return position;

}

/*****************************************************************
   PROGRAM:     getGpsTime
   AUTHOR:      Eric Bachmann, Dave Gay
   DATE:        11 July 1995
   FUNCTION:    Parse the time of a gps message.
   RETURNS:     The time of the gps message in seconds
   CALLED BY:   navPosit (nav.cpp), initializeNavigator(nav.cpp)
   CALLS:       none
*****************************************************************/

double navigatorClass::getGpsTime(const GPSdata& rawMessage)
{
   UNSIGNED_ONEBYTE     tempchar, hours, minutes;
   UNSIGNED_FOURBYTE    tempu4byte;
   double seconds;

   hours   = rawMessage[8];
   minutes = rawMessage[9];

   tempchar         = rawMessage[10];
   tempu4byte       = rawMessage[11];
   tempu4byte       = (tempu4byte<<8) + rawMessage[12];
   tempu4byte       = (tempu4byte<<8) + rawMessage[13];
   tempu4byte       = (tempu4byte<<8) + rawMessage[14];
   seconds = (double)tempchar + (((double)tempu4byte)/1.0E+9);

   return hours * 3600.0 + minutes * 60.0 + seconds;
}

/*****************************************************************
   PROGRAM:    getGpsVelocity
   AUTHOR:     Eric Bachmann, Dave Gay
   DATE:       11 July 1995
   FUNCTION:   Parse the velocity out of a gps message.
   RETURNS:    The velocitiy of the gps message in feet per second
   CALLED BY: navPosit (nav.cpp), initializeNavigator (nav.cpp)
   CALLS:     none
*****************************************************************/

double navigatorClass::getGpsVelocity(const GPSdata& rawMessage)
{
   UNSIGNED_ONEBYTE tempchar=rawMessage[31];

   return (double)(3.2804 * ((tempchar << 8) + rawMessage[32]) /
100.00);
}
// end of file nav.cpp
```

E. GPS.H

```
#ifndef _GPS_H
#define _GPS_H

#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include "toetypes.h"
#include "globals.h"
#include "gpsPort.h"

/******************************************************************
   CLASS:      gpsClass
   AUTHOR:     Eric Bachmann, Dave Gay, Rick Roberts
   DATE:       11 July 1995, last modified January 1997
    FUNCTION:  Reads GPS messages from the GPS buffer. Checks for valid
    checksum and minimun number of satellites in view.
******************************************************************/

class gpsClass {

  public:

    // Class constructor and destructor
    gpsClass() { cerr << "\nconstructing gps1" << endl; };
    ~gpsClass() {}

    // returns the latest gps position and a flag
    Boolean gpsPosition(GPSdata&);

  private:

    // calculates the check sum of the message
    Boolean checkSumCheck(const GPSdata);

};

#endif
```

F. GPS.CPP

```
#include <math.h>
#include "gps.h"

//instantiates serial port communications on comm1,global
//to allow interrupt processing
gpsPortClass port1;

/******************************************************************
   NAME:        gpsPosition
   AUTHOR:      Eric Bachmann, Dave Gay
   DATE:        11 July 1995
   FUNCTION:    Determines if an updated gps position message is
   available and copies it into the input argument 'rawMessage'. If the
   message has a valid checksum and was obtained with atleast three
   satelites in view,  a 'TRUE' is    returned to the caller,
   indicating that the message is valid.
   RETURNS:    TRUE, if a valid position message is contained in the
   input argument.
   CALLED BY:   navPosit (navigator.h)
   CALLS:       Get (buffer.h)        checkSumCheck (gps.h)
******************************************************************/

Boolean gpsClass::gpsPosition(GPSdata& rawMessage)
{
     Boolean validFlag(TRUE);
     unsigned long Mask(4);
     if (port1.Get(rawMessage)) {

      // Check for a valid check sum and 3 or more satelites and DGPS
     if (!checkSumCheck(rawMessage)) {
         cerr << "bad checksum" << endl;
         validFlag = FALSE;
      }
      if (!(rawMessage[39] >= 2)) {
         cerr << "Too few satelites" << endl;
         validFlag = FALSE;
      }

      if (!((rawMessage[GPSBLOCKSIZE - 4]&Mask) == Mask)) {
         cerr << "No DGPS" << endl;
         validFlag = FALSE;
      }

      return validFlag;
   }
   else {
      return FALSE;   // No updated position is available.
   }
}
```

```
/*******************************************************************
   PROGRAM:   checkSumCheck
   AUTHOR:    Eric Bachmann, Dave Gay
   DATE:      11 July 1995
   FUNCTION:  Takes an exclusive or of bytes 2 through 78 in a Motorola
   format(@@EA) position message and compares it to the checksum of the
   message.
   RETURNS:  TRUE,if the message contains a valid checksum
   CALLED BY: gpsPosition (gps)
   CALLS:     none
*******************************************************************/

Boolean gpsClass::checkSumCheck(const GPSdata newMessage)
{
   BYTE chkSum(0);

   for (int i = 2; i < GPSBLOCKSIZE - 3; i++) {
      chkSum ^= newMessage[i];
   }

   return Boolean(chkSum = = newMessage[GPSBLOCKSIZE - 3]);
}
// end of file gps.cpp
```

G. INS.H

```
#ifndef _INS_H
#define _INS_H
#include <time.h>
#include <math.h>
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <fstream.h>
#include <iostream.h>
#include <assert.h>

#include "toetypes.h"
#include "globals.h"
#include "sampler.h"
#include "Matrix.h"

/******************************************************************
   CLASS:      insClass
   AUTHOR:     Eric Bachmann, Dave Gay, Kadir Akyol
   DATE:       11 July 1995 last modified March 1999
   FUNCTION:  Takes in linear accelerations, angular rates, speed and
    heading information and uses Kalman filtering techniques to return a
    dead reconing position.
*******************************************************************/

class insClass {

   public:

      insClass();         // Constructor, initializes gains
      ~insClass() {}      // destructor

      Boolean insPosition(stampedSample&); // returns ins estimated
position

      // Updates the x, y and z of the vehicle posture
      void correctPosition(stampedSample&, double);

      // Sets posture to the origin and develops initial biases
      void insSetUp(double, stampedSample&);

   private:

       Matrix h, h_transpose, p, p_minus, r1, k1, k,  x_hatMinus,
              x_hat, z, i, phi,phi_transpose, q, h1, h1_transpose, k2,
              r2, k3, z3, zMat;

      float posture[6];  // ins estimated posture (x y z phi theta psi)

      double velocities[6]; // ins estimated linear and angular
velocities
       float lastGPStime;   // time of last gps position fix

      int tau;             // filter time constant
      samplerClass sam1;   // sampler instance

      matrix rotationMatrix; // body to euler transformation matrix

      float current[3];    // ins estimated error current
      double biasCorrection[3];  // Software bias corrections for IMU
rate sensors

      // Kalman filter gains.
      float Kone1, Kone2, Ktwo, Kthree1, Kthree2, Kfour1, Kfour2,speed;

      // Transforms body coords to earth coords, removes gravity comp.
      void transformAccels (double[]);

      // Transforms water speed reading to x and y components
      void transformWaterSpeed (double, double[]);

      // Tranforms body euler rates to earth euler rates.
      void transformBodyRates (double[]);

      // Euler integrates the accelerations and updates the velocities
      void updateVelocities (stampedSample&);

      // Euler integrates the velocities and updates the posture
      void updatePosture (stampedSample&);

      // Builds the body to euler rate matrix
      matrix buildBodyRateMatrix();

      // Builds the body to earth rotation matrix
      void buildRotationMatrix();

      // Calculates the imu bias correction during set up
      void calculateBiasCorrections(stampedSample&);

      // Applies bias corrections to a sample
      void applyBiasCorrections(stampedSample&);

      // Reads filter constants from 'ins.cfg'
      void readInsConfigFile();

      //constructs h(4*8) matrix
      void constructHmatrix();

      //constructs P_minus(8*8) matrix
      void constructPminusMatrix();

      //constructs r(4*4) matrix
      void constructR1matrix();

      //constructs h(2*8) matrix (h matrix without GPS)
      void constructH1matrix();

      //constructs r(2*2) matrix (r matrix without GPS)
      void constructR2matrix();

      //constructs phi(8*8) matrix
      void constructPhiMatrix(stampedSample&);

      //constructs q(8*8) matrix
      void constructqMatrix(stampedSample&);
};
// Post multiply a matrix times a vector and return result.
vector operator* (matrix&, double[]);

#endif
```

H. INS.CPP

```
#include <iostream.h>
#include <signal.h>
#include <assert.h>
#include <math.h>

#include "ins.h"

#define SIGFPE 8           // Floating point exception

/******************************************************************
   PROGRAM:  insClass (constructor)
   AUTHOR:   Eric Bachmann,Dave Gay,Rick Roberts, Kadir Akyol
   DATE:     11 July 1995 last modified March 1999
   FUNCTION: Constructor initializes kalman filter gains and  linear
    and angular velocities
    RETURNS:  nothing
   CALLED BY: navigator class
   CALLS:     none
*****************************************************************/

insClass::insClass():h("h matrix",4,8),h_transpose("h transpose", 8,4),
             p_minus("p minus",8,8),r1("r1 matrix",4,4), k1("k1", 4, 4),
             k("k matrix", 8, 4), x_hatMinus("x_hatmin", 8, 1),
             x_hat("x hat", 8,1), i("unit mat", 8, 8),
             phi_transpose("phitranspose", 8, 8), h1("h1",2,8),
             h1_transpose("h1 transpose", 8, 2), r2("r2 matrix",2,2),
             k2("k2", 2, 2), k3("k mat no  gps", 8, 2),
             phi("phi matrix", 8, 8), q("q matrix", 8, 8),
             p("p matrix", 8, 8),z3("z3 matrix",2,1), zMat("zMat",4,1)
{
   cerr << "\nconstructing ins1" << endl;

   readInsConfigFile();             // Read the config file

   constructHmatrix();              //constructs 4*8 h matrix

   constructPminusMatrix();         //constructs 8*8 P_minus matrix

   constructR1matrix();              //constructs 4*4 R1 matrix

   constructH1matrix();              //constructs 2*8 h matrix

   constructR2matrix();              //constructs 2*2 R2 matrix

   velocities[0] = 0.0;              // x dot
   velocities[1] = 0.0;              // y dot
   velocities[2] = 0.0;              // z dot
   velocities[3] = 0.0;              // phi dot
   velocities[4] = 0.0;              // theta dot
   velocities[5] = 0.0;              // psi dot

   posture[0] = 0.0;    // x
   posture[1] = 0.0;    // y
   posture[2] = 0.0;    // z
   posture[3] = 0.0;    // phi
   posture[4] = 0.0;    // theta
   posture[5] = 0.0;    // psi

   cerr << "\nins construction complete" << endl;
}

/******************************************************************
   PROGRAM: insPosit
   AUTHOR:  Eric Bachmann, Dave Gay, Kadir Akyol
   DATE:    11 July 1995 last modified March 1999
    FUNCTION:Make dead reckoning position estimation using kalman
    filtering.Inputs are linear accelerations, angular rates, speed and
    heading.Primary input data is obtained from a sampler object via the
    getSample method. This data is stored in the sample filed of a
    stampedSample structure called newSample. The sample field is then
    used as a working variable as the linear accelerations and angular
    rates it contains are converted to earth coordinates and integrated
    to determine current velocities and posture. The data is
    asynchronous kalman filtered against itself, speed and magnetic
    heading.
    RETURNS:      position in grid coordinates as estimated by the INS
   CALLED BY:     navPosit (nav.cpp)
   CALLS:         getSample (sampler.cpp)
                  findDeltaT (ins.cpp)
                  transformBodyRates (ins.cpp)
                  buildRotationMatrix (ins.cpp)
                  transformAccels (ins)
                  transformWaterSpeed (ins)
******************************************************************/

void fpeInsPosit(int sig)
{if (sig == SIGFPE) cerr << "floating point error in insPosit\n";}

Boolean insClass::insPosition(stampedSample& newSample)
{
   signal (SIGFPE, fpeInsPosit);
    // Working variables
   double thetaA, phiA, xIncline, yIncline;
    // Filter correction for drift and water speed
    double waterSpeedCorrection[3];
   if (sam1.getSample(newSample)) {

      applyBiasCorrections(newSample);

      newSample.rawSample[0] = newSample.sample[0];
      newSample.rawSample[1] = newSample.sample[1];
      newSample.rawSample[2] = newSample.sample[2];
      newSample.rawSample[3] = newSample.sample[3];
      newSample.rawSample[4] = newSample.sample[4];
      newSample.rawSample[5] = newSample.sample[5];
      newSample.rawSample[6] = newSample.sample[6];
      newSample.rawSample[7] = newSample.sample[7];

      xIncline = newSample.sample[0] / GRAVITY;
      yIncline = (newSample.sample[1] -
           (newSample.sample[5] * newSample.sample[6]))
           / (GRAVITY * cos(posture[4]));
```

```
  if (fabs(yIncline) > 1.0) {
      static int inclineCount(0);
      gotoxy(1,24);
      cerr << "Inclination errors: " << ++inclineCount << endl;
      return FALSE;
  }

  // Calculate low freq pitch and roll
  thetaA = asin(xIncline);
  phiA = -asin(yIncline);

  // Transform body rates to euler rates.
  transformBodyRates(newSample.sample);

  // Calculate estimated roll rate (phi-dot).
  velocities[3] = newSample.sample[3] + Kone1 * (phiA - posture[3]);
  // Calculate estimated pitch rate (theta-dot).
  velocities[4] = newSample.sample[4] + Kone2 * (thetaA-posture[4]);
  // Calculate estimated heading rate (psi-dot).
  velocities[5] =
  newSample.sample[5] + Ktwo * (newSample.sample[7] - posture[5]);

  // integrate estimated angular rates to obtain angles
  // pitch rate to angle
  posture[3] += newSample.deltaT * velocities[3];
  // roll rate to angle
  posture[4] += newSample.deltaT * velocities[4];
  // yaw rate to angle
  posture[5] += newSample.deltaT * velocities[5];
  if (newSample.gpsFlag) {

    zMat.copy(0,0,(newSample.sample[6] * cos (posture[5])));
    zMat.copy(1,0,(newSample.sample[6] * sin (posture[5])));
    zMat.copy(2,0,newSample.est.x);
    zMat.copy(3,0,newSample.est.y);

    h.transpose(h_transpose);    //transpose of matrix h
    k1 = (((h*p_minus)*h_transpose)+r1);

    //take inverse of matrix k1
    k1 = k1.invert();

    //calculate matrix k
    k = ((p_minus * h_transpose)* k1);

    //calculate x_hat
    x_hat = ( x_hatMinus + (k * (zMat - (h * x_hatMinus))));

    //calculate I matrix
    i = i.unitMatrix (8);

    p = ((i - (k * h)) * p_minus);     //calculate P matrix
}
else {
   z3.copy(0,0, (newSample.sample[6] * cos (posture[5])));
   z3.copy(1,0, (newSample.sample[6] * sin (posture[5])));
```

```
       //h1 is the h matrix without GPS
       h1.transpose(h1_transpose);

       k2 = (((h1*p_minus)*h1_transpose)+r2);

       k2 = k2.invert();

       //k matrix without gps
       k3 = ((p_minus * h1_transpose)* k2);
       x_hat = ( x_hatMinus + (k3 * (z3 - (h1 * x_hatMinus))));

       i = i.unitMatrix (8);     //calculate I matrix

       p = ((i - (k3 * h1)) * p_minus); //calculate P matrix
  }

  //constructs phi matrix (8*8)
  constructPhiMatrix(newSample);

  //constructs Q matrix (8*8)
  constructqMatrix(newSample);

  //calculate x_hatMinus
  x_hatMinus = ( phi * x_hat );

  //calculate phi_transpose
  phi.transpose(phi_transpose);

  //calculate P_minus
  p_minus = ((( phi * p ) * phi_transpose ) +  q );

  posture[0] += x_hat.getElement(6,0);
  posture[1] += x_hat.getElement(7,0);

  newSample.sample[0] = posture[0] ;
  newSample.sample[1] = posture[1] ;
  newSample.sample[2] = posture[2] ;
  newSample.sample[3] = posture[3];
  newSample.sample[4] = posture[4];
  newSample.sample[5] = posture[5];

  newSample.est.x =  posture[0];
  newSample.est.y =  posture[1];
  newSample.est.z = 0.0 ;

  return TRUE;
  }
  else {
     return FALSE;  // New IMU information was unavailable.
  }
}
```

```
/******************************************************************
   PROGRAM:       insSetUp
   AUTHOR:        Eric Bachmann, Dave Gay
   DATE:          11 July 1995
   FUNCTION:      Initializes the INS system. Sets the posture to the
    origin.Initializes the heading using magnetic compass information.
    Initializes the last GPS fix and last IMU information times.
   RETURNS:       void
   CALLED BY:     initializeNavigator (nav)
   CALLS:         calculateBiasCorrections (ins)
                  getSample (sampler)
                  buildRotationMatrix (ins)
                  transformWaterSpeed (ins)
******************************************************************/

void fpeInsSetUp(int sig)
{if (sig == SIGFPE) cerr << "floating point error in inSetUp\n";}

void insClass::insSetUp(double originTime, stampedSample& posit)
{
   cerr << "   Initializing INS." << endl;
   signal (SIGFPE, fpeInsSetUp);

   sam1.initSampler();        // Initialize the sampler
   sam1.getSample(posit);

   cerr << "   X accel = " << posit.sample[0]
        << ", Y accel = " << posit.sample[1]
        << ", Z accel = " << posit.sample[2] << endl;

   calculateBiasCorrections(posit);     // set imu biases

   posture[5] = posit.sample[7]; //set initial true heading

   buildRotationMatrix();               //set initial speed
   transformWaterSpeed(posit.sample[6], velocities);

   posit.current[0] = current[0];
   posit.current[1] = current[1];
   posit.current[2] = current[2];

   lastGPStime = originTime;            // initialize times

   cerr << "   INS initialization complete." << endl;
}

/**********************************************************
   PROGRAM:   transformAccels
   AUTHOR:    Eric Bachmann, Dave Gay
   DATE:      11 July 1995
   FUNCTION:  Transforms linear accelerations from body coordinates to
    earth coordinates and removes the gravity component in the z
    direction.
   RETURNS:   void
   CALLED BY: navPosit
   CALLS:     none
**********************************************************/
```

```
void insClass::transformAccels (double newSample[])
{
   vector earthAccels;

   newSample[0] -= GRAVITY * sin(posture[4]);
   newSample[1] += GRAVITY * sin(posture[3]) * cos(posture[4]);
   newSample[2] += GRAVITY * cos(posture[3]) * cos(posture[4]);

   earthAccels = rotationMatrix * newSample;

   newSample[0] = earthAccels.element[0];
   newSample[1] = earthAccels.element[1];
   newSample[2] = earthAccels.element[2];
}

/******************************************************************
   PROGRAM:     transformWaterSpeed
   AUTHOR:      Eric Bachmann, Dave Gay
   DATE:        11 July 1995
   FUNCTION:    Transforms water speed into a vector in earth
   coordinates and returns them in the speedCorrection variable.
   RETURNS:     void
   CALLED BY:   navPosit
   CALLS:       none
******************************************************************/

void insClass::transformWaterSpeed (double waterSpeed, double
speedCorrection[])
{
   double water[3] = {waterSpeed, 0.0, 0.0};
   vector waterVelocities = rotationMatrix * water;

   speedCorrection [0] = waterVelocities.element[0];
   speedCorrection [1] = waterVelocities.element[1];
   speedCorrection [2] = waterVelocities.element[2];
}

/******************************************************************
   PROGRAM:    transformBodyRates
   AUTHOR:     Eric Bachmann, Dave Gay
   DATE:       11 July 1995
   FUNCTION:   Tranforms body euler rates to earth euler rates
   RETURNS:    none
   CALLED BY:  insPosit
   CALLS:      buildBodyRateMatrix
******************************************************************/

void insClass::transformBodyRates (double newSample[])
{
   matrix bodyRateMatrix = buildBodyRateMatrix( );
   vector earthRates = bodyRateMatrix * &(newSample[3]);

   newSample[3] = earthRates.element[0];
   newSample[4] = earthRates.element[1];
   newSample[5] = earthRates.element[2];
}
```

```
/***************************************************************
   PROGRAM:   buildBodyRateMatrix
   AUTHOR:    Eric Bachmann, Dave Gay
   DATE:      11 July 1995
   FUNCTION:  Builds body to Euler rate translation matrix.
   RETURNS:   rate translation matrix
   CALLED BY: insPosit
   CALLS:     none
***************************************************************/

matrix insClass::buildBodyRateMatrix()
{
   matrix rateTrans;

   float   tth = tan(posture[4]),
          sphi = sin(posture[3]),
           cphi = cos(posture[3]),
           cth = cos(posture[4]);

   rateTrans.element[0][0] = 1.0;
   rateTrans.element[0][1] = tth * sphi;
   rateTrans.element[0][2] = tth * cphi;
   rateTrans.element[1][0] = 0.0;
   rateTrans.element[1][1] = cphi;
   rateTrans.element[1][2] = -sphi;
   rateTrans.element[2][0] = 0.0;
   rateTrans.element[2][1] = sphi / cth;
   rateTrans.element[2][2] = cphi / cth;

   return rateTrans;
}

/***************************************************************
   PROGRAM:     buildRotationMatrix
   AUTHOR:      Eric Bachmann, Dave Gay
   DATE:        11 July 1995
   FUNCTION:     Sets the body to earth coordinate rotation matrix.
   RETURNS:     void
   CALLED BY:    insPosit, insSetUp
   CALLS:       none
***************************************************************/

void insClass::buildRotationMatrix()
{
   float spsi = sin(posture[5]),
        cpsi = cos(posture[5]),
        sth = sin(posture[4]),
        sphi = sin(posture[3]),
        cphi = cos(posture[3]),
        cth = cos(posture[4]);

   rotationMatrix.element[0][0] = cpsi * cth;
   rotationMatrix.element[0][1] = (cpsi * sth * sphi) - (spsi * cphi);
   rotationMatrix.element[0][2] = (cpsi * sth * cphi) + (spsi * sphi);
   rotationMatrix.element[1][0] = spsi * cth;
   rotationMatrix.element[1][1] = (cpsi * cphi) + (spsi * sth * sphi);
   rotationMatrix.element[1][2] = (spsi * sth * cphi) - (cpsi * sphi);
   rotationMatrix.element[2][0] = -sth;
   rotationMatrix.element[2][1] = cth * sphi;
   rotationMatrix.element[2][2] = cth * cphi;
}

/*****************************************************************
   PROGRAM:    postmultiplication operator *
   AUTHOR:     Eric Bachmann, Dave Gay
   DATE:       11 July 1995
   FUNCTION:   Post multiply a 3 X 3 matrix times a 3 X 1 vector and
   return the result
   RETURNS:  3 X 1 vector
   CALLED BY:
   CALLS:    None1
*****************************************************************/

vector operator* (matrix& transform, double state[])
{
   vector result;

   for (int i = 0; i < 3; i++) {

      result.element[i] = 0.0;

      for (int j = 0; j < 3; j++) {

         result.element[i] += transform.element[i][j] * state[j];
      }
   }
   return result;
}

/*****************************************************************
   PROGRAM:      calculateBiasCorrections
   AUTHOR:       Eric Bachmann, Dave Gay, Rick Roberts
   DATE:         11 July 1995
   FUNCTION:     Calculates the initial imu bias by averaging a number
   of imu readings.
   RETURNS:      none
   CALLED BY:    insSetup
   CALLS:        none
*****************************************************************/

void fpeCalculateBiasCorrections(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
CalculateBiasCorrections\n";}

void insClass::calculateBiasCorrections(stampedSample& biasSample)
{
   signal (SIGFPE, fpeCalculateBiasCorrections);

   int biasNumber(tau/10);

   biasCorrection[0] = 0.0;          // p roll rate
   biasCorrection[1] = 0.0;          // q pitch rate
   biasCorrection[2] = 0.0;          // r yaw rate
   for (int i = 0; i < biasNumber; i++) {

      // Find the average of the biasNumber packets
      while(!sam1.getSample(biasSample)) {/* */};

      // roll-rate/b#
      biasCorrection[0] += biasSample.sample[3]/biasNumber;
      // pitch-rate/b#
      biasCorrection[1] += biasSample.sample[4]/biasNumber;
      // yaw-rate/b#
      biasCorrection[2] += biasSample.sample[5]/biasNumber;
   }

   // set biasSample correction fields to new bias correction values
   // negative biasCorrection value is taken so biases are added to
   // sensor values
   biasSample.bias[0] = biasCorrection[0] = -(biasCorrection[0]);
   biasSample.bias[1] = biasCorrection[1] = -(biasCorrection[1]);
   biasSample.bias[2] = biasCorrection[2] = -(biasCorrection[2]);
}

/*****************************************************************
   PROGRAM:     applyBiasCorrections
   AUTHOR:      Eric Bachmann, Dave Gay, Rick Roberts
   DATE:        11 July 1995
   FUNCTION:    Applies updated bias corrections to a sample.
   RETURNS:     void
   CALLED BY:   insPosit
   CALLS:       none
******************************************************************/

void insClass::applyBiasCorrections(stampedSample& posit)
{
   const float sampleWght(posit.deltaT/tau);
   const float biasWght(1 - sampleWght);

   //Calculate updated bias values
   biasCorrection[0] = (biasWght * biasCorrection[0])
                    - (sampleWght * posit.sample[3]);
   biasCorrection[1] = (biasWght * biasCorrection[1])
                    - (sampleWght * posit.sample[4]);
   biasCorrection[2] = (biasWght * biasCorrection[2])
                    - (sampleWght * posit.sample[5]);

    //Apply the bias to the sample
   posit.sample[3] += biasCorrection[0];
   posit.sample[4] += biasCorrection[1];
   posit.sample[5] += biasCorrection[2];

    //Save the bias to the sample
   posit.bias[0] = biasCorrection[0];
   posit.bias[1] = biasCorrection[1];
   posit.bias[2] = biasCorrection[2];
}
```

```
/*****************************************************************
   PROGRAM:  readInsConfigFile
   AUTHOR:   Rick Roberts, Eric Bachmann
   DATE:     02 Nov 96
   FUNCTION: Reads filter constants from 'ins.cfg'
   RETURNS:  void
   CALLED BY:ins class constructor
   CALLS:    none
*****************************************************************/

void insClass::readInsConfigFile()
{
   cerr << "Reading ins configuration file." << endl;
   ifstream insCfgFile("ins.cfg", ios::in);

   if(!insCfgFile) {
      cerr << "could not open ins configuration file!" << endl;
   }
   else {
      char comment[128];
      insCfgFile
       >> Kone1 >> comment
       >> Kone2 >> comment
       >> Ktwo >> comment
       >> Kthree1 >> comment
       >> Kthree2 >> comment
       >> Kfour1 >> comment
       >> Kfour2 >> comment
       >> tau >> comment
       >> speed >> comment
       >> current[0] >> comment
       >> current[1] >> comment
       >> current[2] >> comment;

      cout << "\nKone1: " << Kone1 << "\nKone2: " << Kone2
           << "\nKtwo:  " << Ktwo << "\nKthree1: " << Kthree1
           << "\nKthree2: " << Kthree2 << "\nKfour1: " << Kfour1
           << "\nKfour2: " << Kfour2 << "\ntau: " << tau
           << "\nx Current: " << current[0] << "\ny Current: "
           << current[1] << "\nz Current: "<< current[2] << endl;
   }

   insCfgFile.close( );
}

/*****************************************************************
   PROGRAM:  constructHmatrix()
   AUTHOR:   Kadir Akyol
   DATE:     01 March 1999
   FUNCTION: constructs h matrix
   RETURNS:  none
   CALLED BY:ins class constructor
   CALLS:    none
*****************************************************************/
```

```
void fpeconstructHmatrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
constructHmatrix\n";}

void insClass::constructHmatrix()
{
   signal (SIGFPE, fpeconstructHmatrix);

   h.copy(0,0,1.0);
   h.copy(1,1,1.0);
   h.copy(2,4,1.0);
   h.copy(2,6,1.0);
   h.copy(3,5,1.0);
   h.copy(3,7,1.0);

   return ;
}//end constructHmatrix()

/******************************************************************
   PROGRAM:    constructPminusMatrix()
   AUTHOR:     Kadir Akyol
   DATE:       01 March 1999
   FUNCTION:   constructs P_minus matrix
   RETURNS:    none
   CALLED BY:  ins class constructor
   CALLS:      none
******************************************************************/

void fpeconstructPminusMatrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
constructPminusMatrix\n";}

void insClass::constructPminusMatrix()
{
   signal (SIGFPE, fpeconstructPminusMatrix);

   p_minus.copy(0,0,0.5);
   p_minus.copy(1,1,0.5);
   p_minus.copy(2,2,1.0);
   p_minus.copy(3,3,1.0);
   p_minus.copy(4,4,3.0);
   p_minus.copy(5,5,3.0);
   p_minus.copy(6,6,5.0);
   p_minus.copy(7,7,5.0);

   return ;
}//end constructPminusMatrix()

/******************************************************************
   PROGRAM:    constructR1matrix()
   AUTHOR:     Kadir Akyol
   DATE:       01 March 1999
   FUNCTION:   constructs r1 matrix
   RETURNS:    none
   CALLED BY:  ins class constructor
   CALLS:      none
******************************************************************/
```

```
void fpeconstructR1matrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
constructR1matrix\n";}

void insClass::constructR1matrix()
{
   signal (SIGFPE, fpeconstructR1matrix);

   r1.copy(0,0,0.5);
   r1.copy(1,1,0.5);

   return ;

}//end constructR1Matrix()

/*******************************************************************
   PROGRAM:    constructH1matrix()
   AUTHOR:     Kadir Akyol
   DATE:       01 March 1999
   FUNCTION:   constructs h matrix
   RETURNS:    none
   CALLED BY:  ins class constructor
   CALLS:      None
*******************************************************************/

void fpeconstructH1matrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
constructH1matrix\n";}

void insClass::constructH1matrix()
{
   signal (SIGFPE, fpeconstructH1matrix);

   h1.copy(0,0,1.0);
   h1.copy(1,1,1.0);

   return ;

}//end constructH1matrix()

/*******************************************************************
   PROGRAM:    constructR2matrix()
   AUTHOR:     Kadir Akyol
   DATE:       01 March 1999
   FUNCTION:   constructs r2 matrix
   RETURNS:    none
   CALLED BY:  ins class constructor
   CALLS:      None
*******************************************************************/

void fpeconstructR2matrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
constructR2matrix\n";}

void insClass::constructR2matrix()
{
   signal (SIGFPE, fpeconstructR2matrix);
   r2.copy(0,0,0.5);
   r2.copy(0,1,0.0);
   r2.copy(1,0,0.0);
   r2.copy(1,1,0.5);

   return ;

}//end constructR2atrix()

/*****************************************************************
   PROGRAM:   constructPhiMatrix()
   AUTHOR:    Kadir Akyol
   DATE:      01 March 1999
   FUNCTION:  constructs phi matrix
   RETURNS:   none
   CALLED BY: insPosit
   CALLS:     None
*****************************************************************/

void fpeconstructPhiMatrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
cunstructPhiMatrix\n";}

void insClass::constructPhiMatrix(stampedSample& delta)
{
   signal (SIGFPE, fpeconstructPhiMatrix);

   double tau_1 = 60.0;
   double tau_3 = 3600.0;

   double xx, yy;

   xx = - (delta.deltaT)/tau_1;
   xx = exp(xx);
   yy = - (delta.deltaT)/tau_3;
   yy = exp(yy);

   phi.copy(0,0,xx);
   phi.copy(1,1,xx);
   phi.copy(2,2,xx);
   phi.copy(3,3,xx);
   phi.copy(4,4,yy);
   phi.copy(5,5,yy);
   phi.copy(6,0,((1-xx)*tau_1));
   phi.copy(6,2,((1-xx)*tau_1));
   phi.copy(7,1,((1-xx)*tau_1));
   phi.copy(7,3,((1-xx)*tau_1));

   return ;

}//end constructPhiMatrix()
```

```
/*****************************************************************
   PROGRAM:    constructqMatrix()
   AUTHOR:     Kadir Akyol
   DATE:       01 March 1999
   FUNCTION:   constructs Q matrix
   RETURNS:    none
   CALLED BY: insPosit
   CALLS:     None
*****************************************************************/

void fpeconstructqMatrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
cunstructqMatrix\n";}

void insClass::constructqMatrix(stampedSample& delt)
{

   signal (SIGFPE, fpeconstructqMatrix);

   double tau_1 = 60.0;
   double tau_3 = 3600.0;
   double zz, ww;


   zz =  -(2.0 * delt.deltaT)/tau_1;
   zz = exp(zz);
   ww =  -(2.0 * delt.deltaT)/tau_3;
   ww = exp(ww);

   q.copy(0,0,((1.0-zz)*(1.0/(2.0*tau_1))));
   q.copy(1,1,((1.0-zz)*(1.0/(2.0*tau_1))));
   q.copy(2,2,((1.0-zz)*(1.0/(2.0*tau_1))));
   q.copy(3,3,((1.0-zz)*(1.0/(2.0*tau_1))));
   q.copy(4,4,((1.0-ww)*(1.0/(2.0*tau_3))));
   q.copy(5,5,((1.0-ww)*(1.0/(2.0*tau_3))));

   return ;

}//end constructqMatrix()

//end of ins.cpp
```

I. SAMPLER.H

```
#ifndef _SAMPLER_H
#define _SAMPLER_H

#include <time.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <fstream.h>
#include <iostream.h>

#include "toetypes.h"
#include "globals.h"
#include "crb.h"
#include "atod.h"
#include "compass.h"

#define MAX_SAMPLE_NUM 1000
#define xyAccelLimit ONE_G          // Max accell in x and y direction
#define zAccelLimit 2 * ONE_G       // Max accel in z direction
#define rateLimit  0.872665         // Max rotational rate in radians
#define speedLimit 25.3             // Max water speed
#define headingLimit 2 * M_PI

const int INBUFFSIZE = 512;

/*********************************************************************
   CLASS:     samplerClass
   AUTHOR:    Eric Bachmann, Dave Gay, Rick Roberts, Kadir Akyol
   DATE:      11 July 1995, last modified March 1999
   FUNCTION:  Formats, timestamps, low pass filters and limit checks
   IMU, water-speed and heading information.
   COMMENTS: This class is extremely dependent upon the specific
   hardware   configuration. It is designed to isolate the INS from
   these particulars.
*********************************************************************/

class samplerClass {

   public:

      samplerClass();               // Class constructor, destructor
      ~samplerClass() {}

      Boolean initSampler();        // Initializes Sampler

      // checks for the arrival of a new sample and formats it
      Boolean getSample(stampedSample&);

   private:

      float pScale;                 // roll
      float qScale;                 // pitch
      float rScale;                 // yaw
      float xAccelScale;                // pitch
      float yAccelScale;                // roll
      float zAccelScale;                // yaw

      float waterSpeedScale;

      float voltage,speed;
      double adOut;

      atodClass ad;

      compassClass comp1;          // instantiate member compass object

      crbClass crossbow1;          // instantiate member a2d object

      long lastImuTime ;

      int subSampleIndex;          // counts channels

      int sampleIndex;             // indexes samples' array

      int sampleCount;             // counts samples

      float samplePeriod;

      Boolean readSamples(stampedSample& newSample);

      void formatSample(stampedSample& newSample);

      void increment(int& index)
      { if (++index == MAX_SAMPLE_NUM) index = 0;}

      void decrement(int& index)
      { if (--index < 0) index = MAX_SAMPLE_NUM - 1;}

      // Reads filter constants from 'sam.cfg'
      void readSamplerConfigFile( );

      double pUnits(double angular)
      { return    (pScale * angular * ((50.0 * 1.5)/32768.0) *
(M_PI/180.0));}

      double qUnits(double angular)
      { return    (qScale * angular * ((50.0 * 1.5)/32768.0) *
(M_PI/180.0));}

      double rUnits(double angular)
      { return    (rScale * angular * ((50.0 * 1.5)/32768.0) *
(M_PI/180.0));}

      double xAccelUnits(double linear)
      { return (xAccelScale *((linear * 2.0 * 1.5 *
GRAVITY)/32768.0));}

      double yAccelUnits(double linear)
      { return (yAccelScale *((linear * 2.0 * 1.5 *
GRAVITY)/32768.0));}
```

```
      double zAccelUnits(double linear)
      { return   (zAccelScale * ((linear * 2.0 * 1.5 *
GRAVITY)/32768.0));}

};
#endif
```

J. SAMPLER.CPP

```
#include "sampler.h"

/***********************************************************************
   PROGRAM:   samplerClass Constructor
   AUTHOR:    Eric Bachmann, Randy Walker, Rick Roberts, Kadir Akyol
   DATE:      12 May 1995, last modified March 1999
   FUNCTION:  Constructs saml, initializes default config values, calls
   readSamplerConfigFile to read any updated values.
   RETURNS:    saml
   CALLED BY: insSetUp (ins.cpp)
   CALLS:     readSamplerConfigFile
***********************************************************************/

samplerClass::samplerClass()
   : sampleIndex(0), subSampleIndex(0),
     pScale(0.0), qScale(0.0), rScale(0.0),
     xAccelScale(0.0), yAccelScale(0.0), zAccelScale(0.0),
     waterSpeedScale(0.0),lastImuTime(0.0)
{
    cerr << "\nconstructing sampler w/ a2d1, comp1" << endl;
    readSamplerConfigFile();
}

/***********************************************************************
   PROGRAM:    initSampler
   AUTHOR:     Eric Bachmann, Randy Walker, Rick Roberts, Kadir Akyol
   DATE:       12 May 1995 last modified March 1999
   FUNCTION:  Instantiates the compass A2D objects.
   RETURNS:    TRUE
   CALLED BY:  insSetUp (ins.cpp)
   CALLS:      initCompass(), AtoD member functions
***********************************************************************/

Boolean samplerClass::initSampler()
{
    sampleIndex = 0;
    subSampleIndex = 0;

    cerr << "      Initializing Sampler" << endl;

    comp1.initCompass();

    cerr << "        Initializing A2D." << endl;

    ad.Initatod(); //ben

    cerr << "         A2D initialization complete." << endl;

    cerr << "      Sampler initialization complete." << endl;

    return TRUE;
}
```

```
/***********************************************************************
   PROGRAM:    getSample
   AUTHOR:     Eric Bachmann, Dave Gay, Kadir Akyol
   DATE:       11 July 1995 last modified March 1999
   FUNCTION:  Prepares raw sample data for use by the INS  object
   RETURNS:   TRUE, if a valid sample was obtained
   CALLED BY: insPosit (ins)     insSetup (ins)
   CALLS:     readSamples (sampler)
              filterSample (sampler)
              formatSample (sampler)
**********************************************************************/

Boolean samplerClass::getSample(stampedSample& newSample)
{
   if (readSamples(newSample)) {  // checks for the arrival of a new
sample

     formatSample(newSample);

     return TRUE;
   }

   return FALSE;                   // Sample packet not available
}

/***********************************************************************
   PROGRAM:  readSamples
   AUTHOR:   Eric Bachmann, Randy Walker
   DATE:     12 May 1996
   FUNCTION: Retrieves all samples of the IMU, water speed, and depth
   that are present in the A2D FIFO until the FIFO is EMPTY. Calculates
   delta_t.
   RETURNS:  TRUE - There were new samples pulled from the FIFO
   FALSE - There were no new samples
   CALLED BY: getSample
   CALLS:     StartConversion(), ConversionDone(), ReadData();
**********************************************************************/

Boolean samplerClass::readSamples(stampedSample& newSample)
{
   if (crossbow1.crbPosition(newSample.crossbowData)) {

      long newImuTime, timeDiff;

      newImuTime =
(newSample.crossbowData[19]*256+newSample.crossbowData[20]);


      if (newImuTime < 0){

         newImuTime += 65536;
      } //end if

      if (lastImuTime != 0){

        if(lastImuTime < newImuTime){
           timeDiff = 65536 - newImuTime + lastImuTime;
        }
        else {

           timeDiff = lastImuTime - newImuTime;
        }

        newSample.deltaT = 0.00000079 * (double)timeDiff;

      }
      else {

        newSample.deltaT = 0.05;
      }

      lastImuTime = newImuTime ;

      //atod converter to read speed voltage
      ad.StartConversion();

      //wait until conversion done
      while (ad.ConversionDone()== 0){};

      //read the converted value
      adOut = ad.ReadData();

      if(adOut>2047){

         adOut = adOut - 4096;
      }//end if

      voltage = (adOut * 0.00244) ;
      newSample.sample[6] = (-7.64/voltage);

      return TRUE;
   }
   else {

      return FALSE;
   }
}

/************************************************************************
   PROGRAM:    formatSample
   AUTHOR:     Eric Bachmann, Dave Gay, Kadir Akyol
   DATE:       11 July 1995 last modified March 1999
   FUNCTION:   Converts integers representing voltage readings into
   real world units which are useable by the INS.
   RETURNS:    void
   CALLED BY:  getSample
   CALLS:      none
************************************************************************/

void samplerClass::formatSample (stampedSample& newSample)
{
   newSample.sample[0] =
  (newSample.crossbowData[11]*256.0+newSample.crossbowData[12]);
```

```
   if( newSample.sample[0] > 32767.0 ){

      newSample.sample[0] -= 65536.0 ;
   }

   newSample.sample[0] =  xAccelUnits(newSample.sample[0]);

   newSample.sample[1] =
(newSample.crossbowData[13]*256.0+newSample.crossbowData[14]);


   if( newSample.sample[1] > 32767.0 ){

      newSample.sample[1] -= 65536.0 ;
   }

   newSample.sample[1] =  yAccelUnits(newSample.sample[1]);

   newSample.sample[2] =
(newSample.crossbowData[15]*256.0+newSample.crossbowData[16]);

   if( newSample.sample[2] > 32767.0 ){

      newSample.sample[2] -= 65536.0 ;
   }

   newSample.sample[2] =  zAccelUnits(newSample.sample[2]);

   newSample.sample[3] =
(newSample.crossbowData[5]*256.0+newSample.crossbowData[6]);

   if( newSample.sample[3] > 32767.0 ){

      newSample.sample[3] -= 65536.0 ;
   }

   newSample.sample[3] =  pUnits(newSample.sample[3]);

   newSample.sample[4] =
(newSample.crossbowData[7]*256.0+newSample.crossbowData[8]);

   if( newSample.sample[4] > 32767.0 ){

      newSample.sample[4] -= 65536.0 ;
   }

   newSample.sample[4] =  qUnits(newSample.sample[4]);

   newSample.sample[5] =
(newSample.crossbowData[9]*256.0+newSample.crossbowData[10]);

   if( newSample.sample[5] > 32767.0 ){

      newSample.sample[5] -= 65536.0 ;
   }
   newSample.sample[5] =  rUnits(newSample.sample[5]);

   newSample.sample[7] = comp1.getHeading();
}

/*********************************************************************
   PROGRAM:    readSamplerConfigFile
   AUTHOR:     Rick Roberts, Eric Bachmann
   DATE:       02 Nov 96
   FUNCTION:   Reads filter constants from 'ins.cfg'
   RETURNS:    void
   CALLED BY:  ins class constructor
   CALLS:      none
   COMMENTS:   * Do not allow blanks in 'comment' section of sam.cfg *
*********************************************************************/

void samplerClass::readSamplerConfigFile()
{
   FILE *samCfgFile;

   if ((samCfgFile = fopen("sam.cfg", "r")) == NULL){
     cerr << "could not open sampler configuration file!" << endl;
   }
   else {
      cerr << "\nReading Sampler configuration file." << endl;

      char line[128];

      fscanf(samCfgFile,"%f%s",&pScale,line);
      cerr << "pScale: " << pScale << endl;

      fscanf(samCfgFile,"%f%s",&qScale,line);
      cerr << "qScale: " << qScale << endl;

      fscanf(samCfgFile,"%f%s",&rScale,line);
      cerr << "rScale: " << rScale << endl;

      fscanf(samCfgFile,"%f%s",&xAccelScale,line);
      cerr << "xAccelScale: " << xAccelScale << endl;

      fscanf(samCfgFile,"%f%s",&yAccelScale,line);
      cerr << "yAccelScale: " << yAccelScale << endl;

      fscanf(samCfgFile,"%f%s",&zAccelScale,line);
      cerr << "zAccelScale: " << zAccelScale << endl;

   }

fclose(samCfgFile);
}
// end of file sampler.cpp
```

K. COMPASS.H

```
#ifndef _MCOMPASS_H
#define _MCOMPASS_H

#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include "toetypes.h"

BYTE asciiToHex(BYTE);          // conversion function prototype

/************************************************************************
   CLASS:     compassClass
   AUTHOR:    Eric Bachmann, Dave Gay, Rick Roberts
   DATE:      11 July 1995, last modified January 1997
   FUNCTION: Reads compass messages from the compass buffer. Checks for
   valid checksum. Corrects heading for magnetic variation. Heading is
   continuous. There is no branch cut at 360 degrees.
************************************************************************/

class compassClass {

  public:

     // class constructor and destructor
     compassClass() : currentHeading(0.0)
     {
        cerr << "Compass constructed." << endl;
     }
     ~compassClass() {}

     float initCompass();                 // initialize currentHeading

     float getHeading();                  // returns the latest heading

  private:
     // Maintains the most recently obtained heading.
     float currentHeading;

     // Maintains the compass headings due to deviation float
     compassHeading[38];

      // calculates the check sum of the message
     Boolean checkSumCheck(const compData);

     // Parses a selected field out of a compass message.
     float parseCompData(const compData, const BYTE);

     // Returns the heading without branch cuts
     float continousHeading(const float);

     // Converts magnetic direction based on magnetic variation.
     float trueHeading(const float);

};
#endif
```

L. COMPASS.CPP

```
#include <math.h>
#include <stdlib.h>
#include "compass.h"
#include "compport.h"

// instantiates serial port communications on comm2, global to allow
// interrupt processing, cleanup to function correctly
compassPortClass port2;

/**********************************************************************
   NAME:      initCompass
   AUTHOR:    Eric Bachmann, Dave Gay, Rick Roberts
   DATE:      11 July 1995
   FUNCTION:  Determines if a valid compass message is held in the
   compass buffer and initializes currentHeading to that value.  Will
   attempt 10 times with a built in delay and then exit with a warning
   if a valid heading is not obtained.
   RETURNS:   currentHeading
   CALLED BY: INSsetUp (ins.cpp)
   CALLS:     Get (buffer.h),  parseCompData (compass.cpp),
              checkSumCheck (gps.h), continuousHeading (compass.cpp),
              trueHeading (compass.cpp)
**********************************************************************/

float compassClass::initCompass()
{
   cerr << "         Initializing Compass" << endl;

   Boolean compFlag(FALSE);
   float tempHeading;
   compData rawMessage;

   // try 10 times to get a valid message
   for (int i = 1 ; ((i < 10) && (compFlag == FALSE)); i++ ) {

      if ((port2.headings.Get(rawMessage)) &&
(checkSumCheck(rawMessage))) {
      tempHeading = parseCompData(rawMessage, 'C') * degToRad;
      currentHeading = continousHeading(trueHeading(tempHeading));
      compFlag = TRUE;
      }
      else {                          // invalid message - delay
      delay(1000);
      }
   }

   if (compFlag == FALSE) {
      cerr << "\nWARNING: UNABLE TO OBTAIN INITIAL COMPASS HEADING!"
   << endl;
      delay(2000);
   }
   else {
      cerr << "         Compass initialization complete." << endl;
   }
return currentHeading;
}

/***********************************************************************
   NAME:       getHeading
   AUTHOR:     Eric Bachmann, Dave Gay, Rick Roberts
   DATE:       11 July 1995
   FUNCTION:   Determines if an updated compass message is available and
   copies it into the input argument 'rawMessage'. If the message has a
   valid checksum, currentHeading is returned to the caller,
   currentHeading is also the default return.
   RETURNS:    currentHeading
   CALLED BY:  navPosit (navigator.h)
   CALLS:      Get (buffer.h)      checkSumCheck (compass.cpp)
***********************************************************************/

float compassClass::getHeading()
{
    float tempHeading;
    compData rawMessage;

    if ((port2.headings.Get(rawMessage))&&(checkSumCheck(rawMessage))) {

      tempHeading = parseCompData(rawMessage, 'C') * degToRad;
      currentHeading = continousHeading(trueHeading(tempHeading));

      return currentHeading;
    }
    else {
      return currentHeading;    // No updated position is available.
    }
}

/***********************************************************************
   NAME:       asciiToHex
   AUTHOR:     Eric Bachmann, Dave Gay
   DATE:       11 July 1995
   FUNCTION:   Administrative conversion function
   RETURNS:    Hex version of an ascii character
   CALLED BY:  checkSumCheck
   CALLS:      None
***********************************************************************/

BYTE asciiToHex(BYTE letter)
{
    if (letter >= 'A') {
       return (letter - 'A' + 10);
    }
    else {
        return (letter - 48);
    }
}
```

```
/**********************************************************************
   PROGRAM:   checkSumCheck
   AUTHOR:    Eric Bachmann, Dave Gay
   DATE:      11 July 1995
   FUNCTION:  Calculates the checksum of the compass message and
   compares it to the indicated checksum of the message.
   RETURNS:   TRUE, if the message contains a valid checksum
   CALLED BY: initCompass, getHeading
   CALLS:     none
**********************************************************************/

Boolean compassClass::checkSumCheck(const compData newMessage)
{
   BYTE calChkSum(0);
   BYTE mesChkSum(0);
   int i;

   for (i = 1; newMessage[i] != '*'; i++) {

      calChkSum ^= newMessage[i];
   }

   mesChkSum = asciiToHex(newMessage[i+1]) * 16
                  + asciiToHex(newMessage[i+2]);

   return Boolean(calChkSum == mesChkSum);
}

/**********************************************************************
   PROGRAM:   trueHeading
   AUTHOR:    Eric Bachmann, Dave Gay
   DATE:      11 July 1995
   FUNCTION:  Converts magnetic direction to true based on local
   magnetic variation.
   RETURNS:   true heading
   CALLED BY: insPosit,  insSetUp
   CALLS:     none
**********************************************************************/

float compassClass::trueHeading(const float magHeading)
{
   static double twoPi(2.0 * M_PI);
   double trueHeading = magHeading + RADIANMAGVAR;

   if (trueHeading > twoPi) {
      trueHeading -= twoPi;
   }

   return trueHeading;
}
```

```
/**********************************************************************
   PROGRAM:     continousHeading
   AUTHOR:      Eric Bachmann
   DATE:        11 July 1995
   FUNCTION:    Maintains track of branch cuts and returns a continous
   heading.
   RETURNS:     continous true heading
   CALLED BY:   insPosit, insSetUp
   CALLS:       none
**********************************************************************/

float compassClass::continousHeading(const float trueHeading)
{
   const float twoPi(2.0 * M_PI);
   static int branchCutCount(0);
   static float previousHeading(trueHeading);

   if ((4.71 < previousHeading) && (trueHeading < 1.57)){
      ++branchCutCount;  // Went through North in a right hand turn
   }
   else {
      if ((1.57 > previousHeading) && (trueHeading > 4.71)) {
       --branchCutCount;    // Went through North in a left hand turn
       }
   }

   previousHeading = trueHeading;

   return trueHeading + (branchCutCount * twoPi);
}

/**********************************************************************
   PROGRAM:      parseCompData
   AUTHOR:       Eric Bachmann
   DATE:         11 July 1995
   FUNCTION:     Parses the heading out of a compass message.
   RETURNS:      the message heading as a float
   CALLED BY:    insPosit, insSetUp
   CALLS:        none
**********************************************************************/

float compassClass::parseCompData(const compData rawMessage, const BYTE
key)
{
   float dataSum(0);
   int j,i;

   for(j = 0; rawMessage[j] != key; j++){}

   j++;

   for(i = 0; rawMessage[i + j] != '.'; i++){}

   switch (i) {

      case 3:
```

```
      dataSum = (rawMessage[j] - 48) * 100.0 +
               (rawMessage[j+1] - 48) * 10.0 +
               (rawMessage[j+2] - 48) + (rawMessage[j+4] - 48) * 0.1;
      break;

      case 2:

      dataSum = (rawMessage[j] - 48) * 10.0 +
               (rawMessage[j+1] - 48) + (rawMessage[j+3] - 48) * 0.1;
      break;

      case 1:

      dataSum = (rawMessage[j] - 48) + (rawMessage[j+2] - 48) * 0.1;

      break;
   }

   return dataSum;
}

// end of file compass.cpp
```

M. CRB.H

```
#ifndef _CRB_H
#define _CRB_H

#include <iostream.h>
#include <fstream.h>
#include <conio.h>

#include "toetypes.h"
#include "globals.h"
#include "crbPort.h"

/**********************************************************************
   CLASS:    crbClass
   AUTHOR:   Kadir Akyol, Erich Bachmann
   DATE:     03 November 1998
   FUNCTION: Reads Crossbow messages from the Crossbow buffer. Checks
   for valid checksum.
**********************************************************************/

class crbClass {

  public:

    // Class constructor and destructor
    crbClass() { cerr << "\nconstructing crossbow" << endl; };
    ~crbClass() {}

    // returns the latest crossbow message
    Boolean crbPosition(CRBdata&);

  private:

    // calculates the check sum of the message
    Boolean checkSumCheck(const CRBdata);

};

#endif
```

N. CRB.CPP

```
#include <math.h>
#include "crb.h"

// instantiates serial port communications on comm3, global to allow
// interrupt processing, cleanup to function properly
crbPortClass port3;

/**********************************************************************
   NAME:        crbPosition
   AUTHOR:      Kadir Akyol, Erich Bachmann
   DATE:        03 November 1998
   FUNCTION:    Determines if an updated crb message is available and
   copies it into the input argument 'rawMessage'. If the message
   has a valid checksum 'TRUE' is returned to the caller, indicating
   that the message is valid.
   RETURNS:     TRUE, if a valid position message is contained in the
                input argument.
   CALLED BY: navPosit (navigator.h)
   CALLS:       Get (buffer.h)
             checkSumCheck (crb.h)
**********************************************************************/

Boolean crbClass::crbPosition(CRBdata& rawMessage)
{
    Boolean validFlag(TRUE);
   //unsigned long Mask(4);

   if (port3.Get(rawMessage)) {

      // Check for a valid check sum
      if (!checkSumCheck(rawMessage)) {
       //cerr << "bad checksum" << endl;
         validFlag = FALSE;
      }//end if

      return validFlag;
   }
   else {
      return FALSE;                // No updated message is available.
   }//end if-else

}//end crbPosition

/**********************************************************************
   PROGRAM:    checkSumCheck
   AUTHOR:     Kadir Akyol, Erich Bachmann
   DATE:       03 November 1998
   FUNCTION:   Adds of bytes 2 through 21 in a Crossbow DMU-VG mode
               messages, compute checksum and compares it to the
               checksum of the message.
   RETURNS:    TRUE, if the message contains a valid checksum
   CALLED BY: crbPosition (gps)
   CALLS:      none
**********************************************************************/
```

```
Boolean crbClass::checkSumCheck(const CRBdata newMessage)
{

  BYTE chkSum(0);

  for (int i = 1; i < CRBBLOCKSIZE - 1; i++) {
      chkSum += newMessage[i];
  }

  chkSum = chkSum % 256;

  return Boolean(chkSum == newMessage[CRBBLOCKSIZE - 1]);

}
// end of file crb.cpp
```

O. MATRIX.H

```
#ifndef __MATRIX_H__
#define __MATRIX_H__

#include <iostream.h>
#include <iomanip.h>

/**********************************************************************
   CLASS:    Matrix
   AUTHOR:   Kadir Akyol, Ildeniz Duman
   DATE:     09 January 1999
   FUNCTION: Executes matrix operations.
**********************************************************************/

class Matrix{

    // overloaded operator<<
    friend ostream &operator<<(ostream &,const Matrix &);

public:

    // default constructor
    Matrix (char * mname="Matrix",int mrow = 4,int mcol = 4);

    //conversition constructor from a two dimensional double array
    Matrix (char * mname,int arrayRow, int arrayCol,double **);


    // destructor
    ~Matrix();

    // copy constructor
    Matrix (const Matrix &);

    // matrix invertion
    Matrix invert();

    // transpose
    void transpose (Matrix &) const;

    // Matrix product
    Matrix operator*(const Matrix &) const;

    // Matrix addition
    Matrix operator+(const Matrix &) const;

    // Matrix subtruction
    Matrix operator-(const Matrix &) const;

    // Matrix assignment
    Matrix &operator=(const Matrix &);

    // Matrix product and assignment
    Matrix &operator*=(const Matrix &);

    // creates a unit matrix
    Matrix unitMatrix(int);

    void copy(int , int ,double );

    // return row no
    int getRow(){return row;}

    // return col no
    int getCol(){return col;}

    // returns an element from the matrix
    double getElement(int i, int j){ return matrix[i][j];}

private:

    // the name of the Matrix
    char * name;

    // the elements of a Matrix
    double ** matrix;

    //row and column
    int row, col;

};

#endif

// end of file Matrix.h
```

P. MATRIX.CPP

```
#include <math.h>
#include <string.h>
#include <assert.h>
#include <signal.h>
#include "Matrix.h"

#define SIGFPE 8

/***********************************************************************
   NAME:      Matrix Constructor
   AUTHOR:    Ildeniz Duman
   DATE:      01 January 1999
   FUNCTION:  Default contructor
   RETURNS:   None
   CALLED BY: insClass (ins.cpp)
   CALLS:     None
***********************************************************************/

Matrix::Matrix (char* mname,int mrow, int mcol)
:row(mrow),col(mcol)
{
    int length = strlen(mname);
    name = new char[length+1];
    assert (name != 0);
    strcpy(name,mname);

    matrix = new double *[row];
    assert (matrix !=0);

    for (int x = 0; x<row; x++){
        matrix[x] = new double [col];
        assert (matrix[x] !=0);
    }
    for (int i = 0;i<row;i++){
        for (int j = 0;j<col;j++){
            matrix[i][j] = 0.0;
        }
    }
}//end Constructor

/***********************************************************************
   NAME:      Matrix Destructor
   AUTHOR:    Ildeniz Duman
   DATE:      01 January 1999
   FUNCTION:  Destructor
   RETURNS:   None
   CALLED BY: None
   CALLS:     None
***********************************************************************/

Matrix::~Matrix()
{
    delete [] name;
    for (int x=0;x<row;x++){
        delete matrix[x];
    }
    delete [] matrix;
}//end destructor

/***********************************************************************
   NAME:       Matrix(Matrix &)
   AUTHOR:     Ildeniz Duman
   DATE:       01 January 1999
   FUNCTION:   Copy contructor
   RETURNS:    None
   CALLED BY:  None
   CALLS:      None
***********************************************************************/

Matrix::Matrix(const Matrix &MAT)
{
    int length = strlen(MAT.name);

    name = new char[length+1];
    assert(name != 0);
    strcpy(name,MAT.name);
    matrix = new double *[MAT.row];
    assert (matrix !=0);
    for (int x = 0; x<MAT.row; x++){
        matrix[x] = new double [MAT.col];
        assert (matrix[x] !=0);
    }
    row=MAT.row;
    col=MAT.col;
    for (int i = 0;i<MAT.row;i++){
        for (int j = 0;j<MAT.col;j++){
            matrix[i][j] = MAT.matrix[i][j];
        }
    }

}// end copy constructor

/***********************************************************************
   NAME:       Matrix()
   AUTHOR:     Ildeniz Duman
   DATE:       01 January 1999
   FUNCTION:   Constructs a matrix from a two dimensional array
   RETURNS:    None
   CALLED BY:  None
   CALLS:      None
***********************************************************************/

Matrix::Matrix (char * mname , int arow, int acol,double ** a)
{
    int length = strlen(mname);

    name = new char[length+1];
    assert (name != 0);
    strcpy(name,mname);
    matrix = new double *[arow];
    assert (matrix !=0);
    for (int x = 0; x<arow; x++){
        matrix[x] = new double [acol];
        assert (matrix[x] !=0);
    }
    row = arow;
    col = acol;
    for (int i = 0;i<row;i++){
        for (int j = 0;j<col;j++){
            matrix[i][j] = a[i][j];
        }
    }

}// end Matrix()

/************************************************************************
   NAME:      operator*()
   AUTHOR:    Kadir Akyol, Ildeniz Duman
   DATE:      01 January 1999
   FUNCTION:  Calculates the Matrix product
   RETURNS:   Matrix
   CALLED BY: insPosit (ins.cpp)
   CALLS:     None
************************************************************************/

void fpeoperatorMul(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
fpeoperatorMul\n";}

Matrix Matrix::operator*(const Matrix &MAT) const
{
    signal (SIGFPE, fpeoperatorMul);

    Matrix dest("Product", row , MAT.col);
    double sum = 0.0f;
    for (int i=0;i<row;i++){
        for (int j=0;j<MAT.col;j++){
            for (int k=0;k<MAT.row;k++){
                sum += matrix[i][k] * MAT.matrix[k][j];
            }
            dest.matrix[i][j]=sum;
            sum = 0.0;
        }

    }

    return(dest);
}//end operator*

/************************************************************************
   NAME:      operator=()
   AUTHOR:    Kadir Akyol, Ildeniz Duman
   DATE:      01 January 1999
   FUNCTION:  Assigns the MAT to current object
   RETURNS:   Matrix &
   CALLED BY: insPosit (ins.cpp)
   CALLS:     None
************************************************************************/
```

```
void fpeoperatorEqual(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
fpeoperatorEqual\n";}

Matrix & Matrix::operator=(const Matrix &MAT)
{
    signal (SIGFPE, fpeoperatorEqual);

    // I let self assingment
    if ((row!=MAT.row) || (col != MAT.col)){
       cout <<"Error in matrix assignment ";
    } else {
        delete [] name;
       int length = strlen(MAT.name);
       name = new char[length+1];
       assert(name != 0);

       for (int i = 0;i<MAT.row;i++){
           for (int j = 0;j<MAT.col;j++){
               matrix[i][j] = MAT.matrix[i][j];
           }
       }
    }
   return (*this);
}//end operator=

/*********************************************************************
   NAME:      unitMatrix()
   AUTHOR:    Kadir Akyol, Ildeniz Duman
   DATE:      01 January 1999
   FUNCTION:  Creates a unit matrix
   RETURNS:   Matrix
   CALLED BY: insPosit (ins.cpp)
   CALLS:     None
*********************************************************************/

Matrix Matrix::unitMatrix (int rowOrCol)
{
    Matrix Unit("unit", rowOrCol , rowOrCol);

    for (int i = 0 ; i<Unit.row ; i++){
       Unit.matrix[i][i] = 1.0;
    }
    return (Unit);
}
// end unitMatrix()

/*********************************************************************
   NAME:      invert()
   AUTHOR:    Kadir Akyol, Ildeniz Duman
   DATE:      01 January 1999
   FUNCTION:  Calculates the matrix inversion
   RETURNS:   Matrix
   CALLED BY: insPosit (ins.cpp)
   CALLS:     None
*********************************************************************/
```

```
void fpeinvert(int sig)
{if (sig == SIGFPE) cerr << "floating point error in fpeinvert\n";}

Matrix Matrix::invert()
{
    signal (SIGFPE, fpeinvert);

    double multiplier=0.0 , divider =0.0;

    Matrix myUnit=myUnit.unitMatrix(row);

    // square matrix check
    if (row == col){

        //inverting the matrix
        for (int j=0; j<col;j++){
            for (int i=0; i<row;i++){
                if (i != j ){
                    multiplier = -matrix[i][j]/matrix[j][j];
                    for (int k=0;k<col;k++){
                        matrix[i][k] += (multiplier * matrix [j][k]);
                        myUnit.matrix[i][k] += (multiplier *
                                          myUnit.matrix[j][k]);
                    }
                }
            }
        }

        // final division to make our matrix a unit matrix
        for (int i = 0 ; i<row ; i++){
            divider = matrix[i][i];
            if (divider != 0.0){
                matrix [i][i] /= matrix [i][i];
                for (int k=0;k<myUnit.row;k++){
                    myUnit.matrix [i][k] /= divider;
                }
            }
        }
    } else {
        cout << "Error : Matrix must be a square matrix "<<endl;
    }
    return (myUnit);
}
// end unitMatrix()

/**********************************************************************
   NAME:      operator*=()
   AUTHOR:    Kadir Akyol, Ildeniz Duman
   DATE:      01 January 1999
   FUNCTION:  Calculates the product and assigns the result to
              current object
   RETURNS:   Matrix
   CALLED BY: None
   CALLS:     None
**********************************************************************/
```

```
void fpeoperatorMulEqual(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
fpeoperatorMulEqual\n";}

Matrix & Matrix::operator*=(const Matrix &MAT)
{
    signal (SIGFPE, fpeoperatorMulEqual);
    *this = *this * MAT;
    return (*this);
}// end operator*=

/*********************************************************************
   NAME:       transpose()
   AUTHOR:     Kadir Akyol, Ildeniz Duman
   DATE:       01 January 1999
   FUNCTION:   Finds the transpose of a matrix
   RETURNS:    None
   CALLED BY:  insPosit (ins.cpp)
   CALLS:      None
*********************************************************************/

void fpetranspose(int sig)
{if (sig == SIGFPE) cerr << "floating point error in fpetranspose\n";}

void Matrix::transpose(Matrix & tr) const
{
    signal (SIGFPE, fpetranspose);
    if ((row == tr.col) && (col == tr.row)){
        for (int i=0;i<row;i++){
            for (int j=0;j<col;j++){
                tr.matrix[j][i] = matrix[i][j];
            }
        }
    }

    return;
}// end transpose()

/*********************************************************************
   NAME:       operator<<()
   AUTHOR:     Kadir Akyol, Ildeniz Duman
   DATE:       01 January 1999
   FUNCTION:   Prints the Matrix in a form, should be written out of
               class
   RETURNS:    ostream object
   CALLED BY:  None
   CALLS:      None
*********************************************************************/

ostream &operator<<(ostream &output, const Matrix &q)
{
    output <<'['<<q.name<<']'<<" "<<q.row<<"x"<<q.col<<endl;;

    for (int k=0;k<q.row;k++){
        for (int m=0;m<q.col;m++){
            output <<"   "<<q.matrix[k][m];
        }
```

```
       output <<endl;
    }

    return output;
}// end operator<<

/**********************************************************************
   NAME:      operator+
   AUTHOR:    Kadir Akyol
   DATE:      01 January 1999
   FUNCTION:  Calculates the Matrix addition
   RETURNS:   Matrix
   CALLED BY: insPosit (ins.cpp)
   CALLS:     None
**********************************************************************/

void fpePlus(int sig)
{if (sig == SIGFPE) cerr << "floating point error in fpePlus\n";}


Matrix Matrix::operator+(const Matrix &MAT) const
{
    signal (SIGFPE, fpePlus);
    Matrix add("Addition", row , col);

   if ((row!=MAT.row) || (col != MAT.col)){
       cout <<"Error in matrix assignment ";
    }
   else {

       for (int i=0;i<row;i++){
           for (int j=0;j<col;j++){

               add.matrix[i][j] = matrix[i][j] + MAT.matrix[i][j];

           }

       }

    }

   return(add);
}//end operator+

/**********************************************************************
   NAME:      operator-
   AUTHOR:    Kadir Akyol
   DATE:      01 January 1999
   FUNCTION:  Calculates the Matrix subtruction
   RETURNS:   Matrix
   CALLED BY: insPosit (ins.cpp)
   CALLS:     None
**********************************************************************/

void fpeMinus(int sig)
{if (sig == SIGFPE) cerr << "floating point error in fpeMinus\n";}
```

```
Matrix Matrix::operator-(const Matrix &MAT) const
{

    signal (SIGFPE, fpeMinus);

    Matrix subt("Subtruction", row , col);

   if ((row!=MAT.row) || (col != MAT.col)){
        cout <<"Error in matrix assignment ";
    }
   else {

        for (int i=0;i<row;i++){
            for (int j=0;j<col;j++){

                subt.matrix[i][j] = matrix[i][j] - MAT.matrix[i][j];

            }

        }

    }

    return(subt);
}//end operator-

/*******************************************************************
   NAME:      copy
   AUTHOR:    Kadir Akyol
   DATE:      01 January 1999
   FUNCTION:  Copies the designeted elementh of Matrix to a Matrix
   RETURNS:   None
   CALLED BY: insPosit (ins.cpp), constructHmatrix (ins.cpp),
          constructPminusMatrix (ins.cpp), constructR1matrix (ins.cpp),
          constructH1matrix (ins.cpp), constructR2matrix (ins.cpp),
          constructPhiMatrix (ins.cpp), constructqMatrix (ins.cpp)
   CALLS:     None
*******************************************************************/

void Matrix::copy(int row, int col,double a)
{

   matrix[row][col] = a;

   return;
}//end of file copy

//end of file Matrix.cpp
```

APPENDIX B: SERIAL COMMUNICATION SOURCE CODE (C++)

A. GLOABAL.H

```
#ifndef _GLOBALS_H
#define _GLOBALS_H

#include <dos.h>

// types
typedef  unsigned char BYTE;
typedef  unsigned short  WORD;
typedef  unsigned long  DWORD;

#define  MEM(seg,ofs)    (*((BYTE far*)MK_FP(seg,ofs)))
#define  MEMW(seg,ofs)   (*((WORD far*)MK_FP(seg,ofs)))

enum Boolean   {FALSE, TRUE};

// basic bit twiddles
#define  set(bit)             (1<<bit)
#define  setb(data,bit)       (data | set(bit))
#define  clrb(data,bit)       (data & !set(bit))
#define  setbit(data,bit)     (data = setb(data,bit))
#define  clrbit(data,bit)     (data = clrb(data,bit))

// specific to ports
#define  setportbit(reg,bit)  (outportb(reg,setb(inportb(reg),bit)))
#define  clrportbit(reg,bit)  (outportb(reg,clrb(inportb(reg),bit)))

// navigation conversion factors and useful global variables
#define MSECS_TO_DEGREES (1.0/(1000.0 * 3600.0))  // time conversion
#define DEGREES_TO_MSECS 3600000.0
#define MINS_TO_MSECS 60000.0

// Conversion constants for location of 36:35:42.2N and 121:52:28.7W
#define LatToFt 0.10134             // converts degrees Latitude to ft
#define LongToFt 0.08156            // converts degrees Longitude to ft
#define HemisphereConversion -1     // -1 if west of of Greenwich

#define RADIANMAGVAR 0.261799    //Local area Magnetic variation in rad

#define radToDeg  (180.0/M_PI)
#define degToRad  (M_PI/180.0)

#endif
```

B. BUFFER.H

```
#ifndef _BUFFER_H
#define _BUFFER_H

#include "toetypes.h"
#include "globals.h"

#define ONE (unsigned short)1

/************************************************************************
   CLASS:         bufferClass
   AUTHOR:        Frank Kelbe, Eric Bachmann, Dave Gay, Rick Roberts
   DATE:          11 July 1995
   FUNCTION:      Base class for use as a polymorphic reference in the
   serial port code which defines a buffer to be used in serial port
   communications.
************************************************************************/

class bufferClass  {

   public:

      bufferClass(WORD sz);   //Constructor
      ~bufferClass() {}

      // Checks for the arrival of new characters in the buffer
      Boolean hasData() { return Boolean(putPtr != getPtr); }

      // How much of the Buffer is used (rounded percentage 0 - 100)
      int capacityUsed();

      Boolean Get(BYTE&);          // read from the buffer
      void Add(BYTE);              // write to the buffer

   protected:

      // Increment the pointer to next position
      void  inc(WORD& index) { if (++index == size) index = 0; }

      WORD before(WORD index)              // decrement the pointer
      { return ((index == 0) ? size - ONE : index - ONE);}

      WORD getPtr;          // Location of unread data
      WORD putPtr;          // Location to read data to
      WORD size;            // Size of the buffer in bytes
      BYTE* buf;
};
#endif
```

C. BUFFER.CPP

```
#include <iostream.h>
#include <stdio.h>
#include "globals.h"
#include "buffer.h"

/*********************************************************************
 FUNCTION NAME:  bufferClass constructor
 AUTHOR:         Frank Kelbe, Eric Bachmann, Dave Gay, Rick Roberts
 DATE:           11 July 1995
 DESCRIPTION:    Instantiates a buffer
 RETURNS:        void
 CALLS:          none
 CALLED BY:      compBuffer, GPSbuffer, bufferedSerialPort constructors
*********************************************************************/

bufferClass::bufferClass(WORD sz) : getPtr(0), putPtr(0), size(sz)
{
   buf = new BYTE[size];
}

/*********************************************************************
 FUNCTION NAME:   capacityUsed()
 AUTHOR:          Frank Kelbe, Eric Bachmann, Dave Gay, Rick Roberts
 DATE:            11 July 1995
 DESCRIPTION:     Returns the rounded percentage of the buffer used.
 RETURNS:         void
 CALLS:           none
 CALLED BY:       bufferedSerialPort::processInterrupt
*********************************************************************/

int bufferClass::capacityUsed()
{
   int cap = (putPtr + size) % size - getPtr;
   return 100 * cap / size;
}

/*********************************************************************
 FUNCTION NAME:  Get
 AUTHOR:         Frank Kelbe, Eric Bachmann, Dave Gay, Rick Roberts
 DATE:           11 July 1995
 DESCRIPTION:    Reads a character from the buffer
 RETURNS:        Boolean
 CALLS:          hasData()
 CALLED BY:      GPSbufferClass, compBufferClass
*********************************************************************/

Boolean bufferClass::Get(BYTE& data)
{
   if (hasData())  {
      data = buf[getPtr];
      inc(getPtr);
      return TRUE;
   }
   return FALSE;
}
```

```
/*********************************************************************
 FUNCTION NAME:  Add
 AUTHOR:         Frank Kelbe, Eric Bachmann, Dave Gay, Rick Roberts
 DATE:           11 July 1995
 DESCRIPTION:    Writes a character to the buffer and checks for buffer
                 overflow
 RETURNS:        void
 CALLS:          hasData
 CALLED BY:      GPSbufferClass, compBufferClass
*********************************************************************/

void bufferClass::Add(BYTE ch)
{
   buf[putPtr] = ch;
   inc(putPtr);

   // if there's no data after adding data, it overflowed
   if (!hasData()) {
      cerr << "\nError: byteBuffer overflow\n";
   }
}
// end of file buffer.cpp
```

D. GPSBUFF.H

```
#ifndef _GPSBUFF_H
#define _GPSBUFF_H

#include "globals.h"
#include "toetypes.h"
#include "buffer.h"

#define  GPSBLOCKS      4
#define  LINE_FEED      10
#define  CARR_RETURN    13

/***************************************************************
  Class buffers GPS position messages via serial port communications.
  Uses a multiple buffer system in which each buffer is capable of
  holding a single position message. Buffers are filled and processed
  sequentially in a round robin fashion. Messages are checked for
  validity only upon attempted reads from the buffer.
***************************************************************/

class gpsBufferClass : public bufferClass  {

   public:

      gpsBufferClass(BYTE GPSblocks = GPSBLOCKS);
      ~gpsBufferClass() { delete [] block; }

      Boolean  hasData();           // a complete structure is ready
      Boolean  Get(BYTE&)   { return FALSE; }
      Boolean  Get(GPSdata);        // fill in a complete structure
      void     Add(BYTE ch);        // build the structure byte by byte

   protected:

      Boolean  validHeader(GPSdata);  // check a block for valid header
      GPSdata  *block;                // hold the buffered GPS data
      WORD     current, last;     // current and last GPS block in use
      BYTE     *putPlace;         // for the next character received
};
#endif
```

E. GPSBUFF.CPP

```
#include <iostream.h>
#include <stdio.h>

#include "gpsbuff.h"

/*********************************************************************
   PROGRAM:       gpsBuffer (Constructor)
   AUTHOR:        Eric Bachmann, Dave Gay
   DATE:          11 July 1995
   FUNCTION:      Allocates message buffers, indicate that no data has
   been received by equalizing current and last and set position into
   which initial character will be read.
   RETURNS:   nothing.
   CALLED BY:     navigator class (nav.h)
   CALLS:         none.
*********************************************************************/

gpsBufferClass::gpsBufferClass(BYTE GPSblocks) : current(0), last(0),
                 bufferClass(GPSblocks) // Call to base class
constructor
{
   cerr << "constructing gpsBuffer" << endl;
   block = new GPSdata[GPSblocks];//Create an array of GPSdata elements
   putPlace = &(block[current][0]);  // Set the place for the first
character
}

/*********************************************************************
   PROGRAM:    Add
   AUTHOR:     Eric Bachmann, Dave Gay
   DATE:       11 July 1995
   FUNCTION:   Interrupt driven routine which writes incoming
   characters into the gps buffers
   RETURNS:    nothing.
   CALLED BY:  interupt driven by bufferedSerialPort
   CALLS:      none.
*********************************************************************/

void gpsBufferClass::Add(BYTE data)
{
   static BYTE lastChar(data);    // Holds last for <cr> <lf> detection
   static Boolean lfFlag = FALSE; // True when message end is detected

   if (lfFlag && (data == '@')) {       // Is a new message starting?
      last = current;         // Set last to buffer with newest message.
      inc(current);                 // Set current to the next buffer
      // Set putPlace to the beginning of the next buffer.
      putPlace = &(block[current][0]);
      lfFlag = FALSE;               // reset for end of next message.
   }

   *putPlace++ = data;              // Write character into the buffer.

   //Has the end of a message been received?
   if ((lastChar == CARR_RETURN) && (data == LINE_FEED)) {
      lfFlag = TRUE;
   }
   lastChar = data;       //Save last character for <cr> <lf> detection
}

/**********************************************************************
   PROGRAM:       Get
   AUTHOR:        Eric Bachmann, Dave Gay
   DATE:          11 July 1995
   FUNCTION:      Checks to see if a new message has arrived, copies it
   into the input argument data and returns a flag to indicate whether
   a new message was received
   RETURNS:       TRUE, if a new valid position has been received.
                  FALSE, otherwise
   CALLED BY:     navPosit (nav.cpp),  initializeNavigator (nav.cpp)
   CALLS:         gpsBufferClass::hasData
**********************************************************************/

Boolean gpsBufferClass::Get(GPSdata data)
{
   if (hasData( ))  {         // Has a new valid message been received.
      // Copy the message out of the buffer.
      memcpy (data, block + last, GPSBLOCKSIZE);
      last = current;      // Indicate that this message has been read.
      return TRUE;
   }
   else {
       return FALSE;
   }
}

/**********************************************************************
   PROGRAM:   hasData
   AUTHOR:    Eric Bachmann, Dave Gay
   DATE:      11 July 1995
   FUNCTION:  Determines whether a new message has been received and
   checks to see if it has a valid header.
   RETURNS:   TRUE, if a new valid message has been received.
   CALLED BY: gpsBufferClass::Get (buffer.cpp)
   CALLS:     validHeader (buffer.cpp)
**********************************************************************/

Boolean gpsBufferClass::hasData( )
{
    // Has a new message with a valid header been received
    if (last != current) {
       if (validHeader(block[last])) {
          return TRUE;
       }
       else {
          return FALSE;
      }
   }
   return FALSE;
}
```

```
/*********************************************************************
   PROGRAM:    validHeader
   AUTHOR:     Eric Bachmann, Dave Gay
   DATE:       11 July 1995
   FUNCTION:   Checks to see if a message has the proper header for a
   Motorola position message. (@@Ea)
   RETURNS:    TRUE, if the header is valid. FALSE, otherwise.
   CALLED BY:  gpsBufferClass::hasData (buffer.cpp)
   CALLS:      none.
*********************************************************************/

Boolean gpsBufferClass::validHeader(GPSdata dataPtr)
{
   if ((dataPtr[0] = = '@') && (dataPtr[1] = = '@') &&
       (dataPtr[2] = = 'E') && (dataPtr[3] = = 'a'))  {
   ;
      return TRUE;
   }
   else {
      return FALSE;
  }
}
// end of file gpsbuff.cpp
```

F. COMPBUFF.H

```
#ifndef __COMPBUFF_H
#define __COMPBUFF_H

#include "toetypes.h"
#include "globals.h"
#include "buffer.h"

#define  COMPBLOCKS          8
#define  LINE_FEED           10
#define  CARR_RETURN         13
#define  g             103
#define  o             111

/*********************************************************************
  Class buffers COMPASS messages received via serial port
communications. Uses a multiple buffer system in which each buffer is
capable of holding a single message. Buffers are filled and processed
sequentially in a round robin fashion. Messages are checked for
validity only upon attempted reads from the buffer.
*********************************************************************/

class compBufferClass : public bufferClass  {

public:

   compBufferClass(BYTE compBlocks = COMPBLOCKS);

   ~compBufferClass() {delete [] block;}

   Boolean  hasData();                // a complete structure is ready
   Boolean  Get(BYTE&) {return FALSE;} // satisfy inheritance
requirements
   Boolean  Get(compData);       // get a complete structure filled in
   void     Add(BYTE ch);        // build the structure byte by byte

protected:                              // for inheritance

   Boolean  validHeader(compData);   // check a block for valid header
   compData *block;                  // points to array of compass msgs
   WORD     current, last;       // current and last comp block in use

   BYTE     *putPlace;           // for the next character received
};

#endif
```

G. COMPBUFF.CPP

```
#include <iostream.h>
#include <stdio.h>

#include "compbuff.h"

/*********************************************************************
   PROGRAM:   compBuffer (Constructor)
   AUTHOR:    Eric Bachmann, Randy Walker
   DATE:      28 April 1996
   FUNCTION:  Allocates message buffers, indicates that no data has
   been received by equalizing current and last and sets the position
   into which initial character will be read.
   RETURNS:   nothing.
   CALLED BY: compassClass (compass.h)
   CALLS:     none.
*********************************************************************/

compBufferClass::compBufferClass(BYTE compBlocks):  current(0),
last(0), bufferClass(compBlocks) // Call to base class constructor
{
      cerr << "compBuffer constructor called" << endl;
      block = new compData[compBlocks]; // Create array of message
buffers
      putPlace = &(block[current][0]);  // Set position for first char

      cerr << "compBuffer constructed." << endl;
}

/*********************************************************************
   PROGRAM:   compBuffer::Add
   AUTHOR:    Eric Bachmann, Randy Walker
   DATE:      28 April 1996
   FUNCTION:  Interrupt driven routine which writes incoming characters
   into the compass message buffers
   RETURNS:   nothing.
   CALLED BY: interrupt driven by compassPort
   CALLS:     none.
*********************************************************************/

void compBufferClass::Add(BYTE data){

   static Boolean lfFlag = FALSE; //True, if message end detected
   static int messageCount(0);  // Counts characters in current message

   if (lfFlag && (data == ’$’)) {         // Is a new message starting?

      last = current;        // Set last to buffer with newest message.
      inc(current);          // Set current to the next buffer

      // Set putPlace to the beginning of the next buffer.
      putPlace = &(block[current][0]);
      lfFlag = FALSE;               // reset for end of next message.
   }

   *putPlace++ = data;              // Write character into the buffer.
   messageCount++;

   //Has the end of a message been received (<cr><lf>)?
   if (data == LINE_FEED) {
      lfFlag = TRUE;
   }
}

/**********************************************************************
   PROGRAM:   compBuffer::Get
   AUTHOR:    Eric Bachmann, Randy Walker
   DATE:      28 April 1996
   FUNCTION:  Checks to see if a new message has arrived, copies it
   into the input argument data and returns a flag to indicate whether
   a new message was received.
   RETURNS:   TRUE, if a new valid position has been received. FALSE,
   otherwise
   CALLED BY: compass.cpp
   CALLS:     compBuffer::hasData
**********************************************************************/

Boolean compBufferClass::Get(compData data)
{
   if (hasData( ))  {      // Has a new valid message been received.
      // Copy the message out of the buffer.
      memcpy (data, block + last, COMPSIZE);
      last = current;      // Indicate that this message has been read.
      return TRUE;
   }
   else {
      return FALSE;
   }
}

/**********************************************************************
   PROGRAM:   compBuffer::hasData
   AUTHOR:    Eric Bachmann, Randy Walker
   DATE:      28 April 1996
   FUNCTION:  Determines whether a new message has been received and
   Checks to see if it has a valid header.
   RETURNS:   TRUE, if a new valid message has been received.
   CALLED BY: compBuffer::Get
   CALLS:     validHeader (compBuffer.cpp)
**********************************************************************/

Boolean compBufferClass::hasData()
{
   if ((last != current) && (validHeader(block[last]))) {
      return TRUE;
   }
   else {
      return FALSE;
   }
}
```

```
/**********************************************************************
   PROGRAM:     validHeader
   AUTHOR:      Eric Bachmann, Dave Gay
   DATE:        11 July 1995
   FUNCTION:    Checks to see if a message has the proper header for a
   compass message. ($C)
   RETURNS:     TRUE, if the header is valid. FALSE, otherwise.
   CALLED BY:   compBuffer::hasData
   CALLS:       none.
**********************************************************************/

Boolean compBufferClass::validHeader(compData dataPtr)
{
  if ((dataPtr[0] == '$') && (dataPtr[1] == 'C')) {
     return TRUE;
  }
  else {
     return FALSE;
  }
}
//end of file compbuff.cpp
```

H. CRBBUFF.H

```
#ifndef _CRBBUFF_H
#define _CRBBUFF_H

#include "globals.h"
#include "toetypes.h"
#include "buffer.h"

#define  CRBBLOCKS      10
#define  LINE_FEED      10
#define  CARR_RETURN    13


/*********************************************************************
    Class buffers Crossbow messages via serial port communications.
 Uses a multiple buffer system in which each buffer is capable of
 holding a single message. Buffers are filled and processed sequentially
 in a round robin fashion. Messages are checked for validity only upon
 attempted reads from the buffer.
*********************************************************************/

class crbBufferClass : public bufferClass  {

   public:

      crbBufferClass(BYTE CRBblocks = CRBBLOCKS);
      ~crbBufferClass() { delete [] block; }

      Boolean  hasData();          // a complete structure is ready
      Boolean  Get(BYTE&)    { return FALSE; }
      Boolean  Get(CRBdata);       // fill in a complete structure
      void     Add(BYTE ch);       // build the structure byte by byte

   protected:

      Boolean  validHeader(CRBdata); // check a block for valid header
      CRBdata  *block;               // hold the buffered Crossbow data
      WORD     current, last; // current and last Crossbow block in use
      BYTE     *putPlace;     // for the next character received
};

#endif
```

I. CRBBUFF.CPP

```
#include <iostream.h>
#include <stdio.h>

#include "crbbuff.h"

/*********************************************************************
   PROGRAM:   crbBuffer (Constructor)
   AUTHOR:    Kadir Akyol, Eric Bachmann
   DATE:      03 November 1998
   FUNCTION:  Allocates message buffers, indicate that no data has been
   received by equalizing current and last and set position into which
   initial character will be read.
   RETURNS:   nothing.
   CALLED BY: none
   CALLS:     none.
*********************************************************************/

crbBufferClass::crbBufferClass(BYTE CRBblocks) : current(0), last(0),
               bufferClass(CRBblocks) // Call to base class constructor
{
   cerr << "constructing crossbow buffer" << endl;
   block = new CRBdata[CRBblocks];//Create an array of CRBdata elements
   putPlace = &(block[current][0] // Set the place for the first char
}

/*********************************************************************
   PROGRAM:   Add
   AUTHOR:    Kadir Akyol, Eric Bachmann
   DATE:      03 November 1998
   FUNCTION:  Interrupt driven routine which writes incoming characters
   into the crossbow message buffers.
   RETURNS:   nothing.
   CALLED BY: interupt driven by bufferedSerialPort
   CALLS:     none.
********************************************************************/

void crbBufferClass::Add(BYTE data)
{
    static short byteCount(22);
    byteCount++;

    if (data = = 0xFF && (byteCount > 22)) {

       last = current;      // Set last to buffer with newest message.
       inc(current);                // Set current to the next buffer
       byteCount=1;
       // Set putPlace to the beginning of the next buffer.
       putPlace = &(block[current][0]);
   }//end if

   *putPlace++ = data;              // Write character into the buffer.
}
```

```
/***********************************************************************
   PROGRAM:   Get
   AUTHOR:    Kad5ir Akyol, Eric Bachmann
   DATE:      03 November 1998
   FUNCTION:  Checks to see if a new message has arrived, copies it
   into the input argument data and returns a flag to indicate whether a
   new message was received       RETURNS:   TRUE, if a new valid
   position has been received. FALSE, otherwise
   CALLED BY: crb.cpp
   CALLS:     crbBufferClass::hasData
***********************************************************************/

Boolean crbBufferClass::Get(CRBdata data)
{
   if (hasData())  {        // Has a new valid message been received.
      memcpy (data, block + last, CRBBLOCKSIZE);
      last = current;       // Indicate that this message has been read.
      return TRUE;
   }
   else {
      return FALSE;
   }
}

/***********************************************************************
   PROGRAM:   hasData
   AUTHOR:    Kadir Akyol, Eric Bachmann
   DATE:      03 November 1998
   FUNCTION:  Determines whether a new message has been received and
   Checks to see if it has a valid header.
   RETURNS:   TRUE, if a new valid message has been received.
   CALLED BY: crbBufferClass::Get (buffer.cpp)
   CALLS:     validHeader (crbbuffer.cpp)
***********************************************************************/

Boolean crbBufferClass::hasData( )
{
   if (last != current) {
      if (validHeader(block[last])) {
          return TRUE;
      }
      else {
          return FALSE;
      }
   }
   return FALSE;
}

/***********************************************************************
   PROGRAM:   validHeader
   AUTHOR:    Kadir Akyol, Eric Bachmann
   DATE:      03 November 1998
   FUNCTION:  Checks to see if a message has the proper header.
   RETURNS:   TRUE, if the header is valid. FALSE, otherwise.
   CALLED BY: crbBufferClass::hasData
   CALLS:     none.
***********************************************************************/
```

```
Boolean crbBufferClass::validHeader(CRBdata dataPtr)
{
   if ((dataPtr[0] == 0xff)){
       return TRUE;
   }
   else {
       return FALSE;
   }
}
// end of file crbbuff.cpp
```

J. GPSPORT.H

```
#ifndef _GPSPORT_H
#define _GPSPORT_H

#include <dos.h>
#include <stdio.h>
#include "toetypes.h"
#include "globals.h"
#include "serial.h"
#include "gpsbuff.h"

// this is the type for a standard interrupt handler
typedef void interrupt (IntHandlerType)(...);

// com handler to interface with processInterrupt
void interrupt COM1handler(...);

/*********************************************************************
   CLASS:   gpsPortClass
   AUTHOR:  Rick Roberts
   DATE:    28 January 1997
   FUNCTION:Defines a buffered serial port which is interrupt driven
   on receive, and buffers all incoming characters in the gps buffer
*********************************************************************/

class gpsPortClass : public serialPortClass  {

   public:

      gpsPortClass(COMport portnum = COM1, BYTE irq = 4,
                   BaudRate speed = b9600,
                   ParityType parity = NOPARITY, BYTE wordlen = 8,
                   BYTE stopbits = 1, handShake hs = XON_XOFF);
      ~gpsPortClass();

      Boolean Get(GPSdata& data);     // buffered version
      void  processInterrupt();       // buffers the incoming character

   protected:

      gpsBufferClass messages;

      BYTE irqbit; // Value to allow enable PIC interrupts for COM port
      BYTE origirq;    // keep the original 8259 mask register value
      BYTE comint;

      IntHandlerType *origcomint; // keep original vector for restoring

      // this allows the actual handler to access processInterrupt()
      friend IntHandlerType COM2handler;

};
extern gpsPortClass port1;

#endif
```

K. GPSPORT.CPP

```
#include <iostream.h>
#include <stdio.h>

#include "gpsPort.h"

/************************************************************************
   PROGRAM:   gpsPortClass (Constructor)
   AUTHOR:    Rick Roberts
   DATE:      28 January 1997
   FUNCTION:  Initializes the interrupts for the gps Serial Port.
             1) takes over the original COM interrupt
             2) sets the port bits, parity, and stop bit
             3) enables interrupts on the 8250 (async chip)
             4) enables the async interrupt on the 8259 PIC
************************************************************************/

gpsPortClass::gpsPortClass(COMport portnum, BYTE irq, BaudRate baud,
                        ParityType parity, BYTE wordlen,
                        BYTE  stopbits, handShake hs)    :
                        serialPortClass(portnum, baud, parity, wordlen,
                        stopbits, hs), irqbit(irq), comint(irqbit+8)
{
   cerr << "gpsPort constructor called" << endl;

   if (ShakeType == RTS_CTS)  {  // turn it off first, because it was
enabled
        setDTRoff();             // in the base class
        setRTSoff();
   }

   origcomint = getvect(comint);      // remember the original vector

   setvect(comint,COM1handler);       // point to the new handler

   setportbit(MCR,3);           // turn OUT2 on
   disable();           // disable all interrupts - critical section
   setportbit(IER,rx_rdy);      // enable ints on receive only
   origirq = inportb(IRQPORT);  // remember how it was
   clrportbit(IRQPORT,irqbit);  // enable COM ints

   if (ShakeType == RTS_CTS)  {
        setDTRon();
        setRTSon();
   }
   enable();

   EOI;
   cerr << "exiting gpsPort constructor" << endl;
}
```

```
/*******************************************************************
   PROGRAM:    ~gpsPortClass
   AUTHOR:     Rick Roberts, Frank Kelbe, Eric Bachmann, Dave Gay
   DATE:       28 January 1997
   FUNCTION:   Resets the interrupts.
              1) disables the 8250 (async chip)
              2) disables the interrupt chip for async int
              3) resets the 8259 PIC
*******************************************************************/

gpsPortClass::~gpsPortClass()
{
   setvect(comint,origcomint);   // set the interrupt vector back
   outportb(IER,0);              // disable further UART interrupts
   outportb(MCR,0);              // turn everything off
   outportb(IRQPORT,origirq);
   EOI;
}

/*******************************************************************
   PROGRAM:    Get
   AUTHOR:     Frank Kelbe, Eric Bachmann, Dave Gay
   DATE:       11 July 1995
   FUNCTION:   Calls Get based on buffer type
*******************************************************************/

Boolean gpsPortClass::Get(GPSdata& data)
{
   return messages.Get(data);
}

/*******************************************************************
   PROGRAM: COM1handler
   AUTHOR:  Frank Kelbe, Eric Bachmann, Dave Gay, Rick Roberts
   DATE:    11 July 1995, last modified January 1997
   FUNCTION:Specific interrupt handler which maps each interrupt to
   the proper ISR.
*******************************************************************/

void interrupt COM1handler(...)
{
   port1.processInterrupt();
   EOI;
}

/*******************************************************************
   PROGRAM:    processInterrupt
   AUTHOR:     Frank Kelbe, Eric Bachmann, Dave Gay, Rick Roberts
   DATE:       11 July 1995
   FUNCTION:   Calls the ISR based upon buffer type
*******************************************************************/

void gpsPortClass::processInterrupt()
{
   if (dataReady())  {               // make sure there's a char there
        BYTE data = inportb(RX);       // read character from 8250
        messages.Add(data);
        if (ShakeType == RTS_CTS && messages.capacityUsed() >
ALMOST_FULL)
             setDTRoff();
   }
}
// end of file gpsport.cpp
```

L. COMPPORT.H

```
#ifndef _MCOMPORT_H
#define _MCOMPORT_H

#include <dos.h>
#include <stdio.h>

#include "toetypes.h"
#include "globals.h"
#include "serial.h"
#include "compbuff.h"

// this is the type for a standard interrupt handler
typedef void interrupt (IntHandlerType)(...);

// com handler to interface with processInterrupt
void interrupt COM2handler(...);

/*********************************************************************
   CLASS:   compassPortClass
   AUTHOR:  Rick Roberts
   DATE:    28 January 1997
   FUNCTION:    Defines a buffered serial port which is interrupt
   driven on receive, and buffers all incoming characters in the
   compass buffer
**********************************************************************/

class compassPortClass : public serialPortClass  {

   friend compassClass;

   public:

      compassPortClass(COMport portnum = COM2, BYTE irq = 3,
                   BaudRate speed = b9600,
                   ParityType parity = NOPARITY, BYTE wordlen = 8,
                   BYTE stopbits = 1, handShake hs = NONE);

      ~compassPortClass();

      Boolean     Get(BYTE& data);  // buffered version

      void processInterrupt();      // buffers the incoming character

   private:

      compBufferClass headings;

      BYTE irqbit; // Value to allow enable PIC interrupts for COM port
      BYTE origirq;    // keep the original 8259 mask register value
      BYTE comint;

      IntHandlerType *origcomint; // keep original vector for restoring

      // this allows the actual handler to access processInterrupt()
```

```
	friend IntHandlerType COM2handler;
};

extern compassPortClass port2;

#endif
```

M. COMPPORT.CPP

```
#include <iostream.h>
#include "compport.h"

/**********************************************************************
   PROGRAM:   compassPortClass (Constructor)
   AUTHOR:    Rick Roberts
   DATE:      28 January 1997
   FUNCTION:  Initializes the interrupts for the compass Serial Port.
             1) takes over the original COM interrupt
             2) sets the port bits, parity, and stop bit
             3) enables interrupts on the 8250 (async chip)
             4) enables the async interrupt on the 8259 PIC
**********************************************************************/

compassPortClass::compassPortClass(COMport portnum, BYTE irq,
                  BaudRate baud, ParityType parity, BYTE wordlen,
                  BYTE stopbits, handShake hs) :
                  serialPortClass(portnum, baud, parity, wordlen,
                  stopbits, hs)
{
   cerr << "compassPort constructor called" << endl;

   irqbit = irq;
   comint = irqbit + 8;

   if (ShakeType == RTS_CTS)  { // turn it off first, because it was
enabled
        setDTRoff();            // in the base class
        setRTSoff();
   }

   origcomint = getvect(comint);       // remember the original vector

   setvect(comint,COM2handler);            // point to the new handler

   setportbit(MCR,3);           // turn OUT2 on
   disable();             // disable all interrupts - critical section
   setportbit(IER,rx_rdy);      // enable ints on receive only
   origirq = inportb(IRQPORT);  // remember how it was
   clrportbit(IRQPORT,irqbit);  // enable COM ints

   if (ShakeType == RTS_CTS)  {
        setDTRon();
        setRTSon();
   }
   enable();

   EOI;
   cerr << "exiting compassPort constructor" << endl;
}
```

```
/**********************************************************************
   PROGRAM:   ~compassPort
   AUTHOR:    Rick Roberts, Frank Kelbe, Eric Bachmann, Dave Gay
   DATE:      28 January 1997
   FUNCTION:  Resets the interrupts.
             1) disables the 8250 (async chip)
             2) disables the interrupt chip for async int
             3) resets the 8259 PIC
**********************************************************************/

compassPortClass::~compassPortClass()
{
   setvect(comint,origcomint);   // set the interrupt vector back
   outportb(IER,0);              // disable further UART interrupts
   outportb(MCR,0);              // turn everything off
   outportb(IRQPORT,origirq);
   EOI;
}

/**********************************************************************
   PROGRAM:   Get
   AUTHOR:    Frank Kelbe, Eric Bachmann, Dave Gay
   DATE:      11 July 1995
   FUNCTION:  Calls Get based on buffer type
*********************************************************************/

Boolean compassPortClass::Get(BYTE& data)
{
   return headings.Get(data);
}

/**********************************************************************
   PROGRAM: COM2handler
   AUTHOR:  Frank Kelbe, Eric Bachmann, Dave Gay, Rick Roberts
   DATE:    11 July 1995, last modified January 1997
   FUNCTION:Specific interrupt handler which maps each interrupt to
   the proper ISR.
**********************************************************************/

void interrupt COM2handler(...)
{
   port2.processInterrupt();
   EOI;
}

/**********************************************************************
   PROGRAM:   processInterrupt
   AUTHOR:    Frank Kelbe, Eric Bachmann, Dave Gay, Rick Roberts
   DATE:      11 July 1995
   FUNCTION:  Calls the ISR based upon buffer type
**********************************************************************/

void compassPortClass::processInterrupt()
{
   if (dataReady())  {              // make sure there's a char there
        BYTE data = inportb(RX);    // read character from 8250
        headings.Add(data);
        if (ShakeType == RTS_CTS && headings.capacityUsed() >
ALMOST_FULL)
             setDTRoff();
   }
}
// end of file compport.cpp
```

N. CRBPORT.H

```
#ifndef _CRBPORT_H
#define _CRBPORT_H

#include <dos.h>
#include <stdio.h>
#include "toetypes.h"
#include "globals.h"
#include "serial.h"
#include "crbbuff.h"

// this is the type for a standard interrupt handler
typedef void interrupt (IntHandlerType)(...);

// com handler to interface with processInterrupt
void interrupt COM3handler(...);

/*********************************************************************
   CLASS:   crbPortClass
   AUTHOR:  Kadir Akyol, Erich,Bachmann
   DATE:    03 November 1998
   FUNCTION:Defines a buffered serial port which is interrupt
   driven on receive, and buffers all incoming characters in the
   gps buffer
*********************************************************************/

class crbPortClass : public serialPortClass  {

   public:

      crbPortClass(COMport portnum = COM3, BYTE irq = 5,
                   BaudRate speed = b38400,
                   ParityType parity = NOPARITY, BYTE wordlen = 8,
                   BYTE stopbits = 1, handShake hs = NONE);
      ~crbPortClass();

      Boolean Get(CRBdata& data);     // buffered version
      void  processInterrupt();       // buffers the incoming character


   protected:

      crbBufferClass messages;

      BYTE irqbit; // Value to allow enable PIC interrupts for COM port
      BYTE origirq;    // keep the original 8259 mask register value
      BYTE comint;

      IntHandlerType *origcomint; // keep original vector for restoring

      // this allows the actual handler to access processInterrupt()
      friend IntHandlerType COM2handler;
};
extern crbPortClass port3;

#endif
```

O. CRBPORT.CPP

```
#include <iostream.h>
#include <stdio.h>
#include "crbPort.h"

/**********************************************************************
   PROGRAM:   crbPortClass (Constructor)
   AUTHOR:    Kadir Akyol, Eric Bachmann
   DATE:      03 November 1998
   FUNCTION:  Initializes the interrupts for the gps Serial Port.
             1) takes over the original COM interrupt
             2) sets the port bits, parity, and stop bit
             3) enables interrupts on the 8250 (async chip)
             4) enables the async interrupt on the 8259 PIC
**********************************************************************/

crbPortClass::crbPortClass(COMport portnum, BYTE irq, BaudRate baud,
                        ParityType parity, BYTE wordlen,
                        BYTE  stopbits, handShake hs)    :
                        serialPortClass(portnum, baud, parity, wordlen,
                        stopbits, hs), irqbit(irq), comint(irqbit+8)
{
   cerr << "crbPort constructor called" << endl;

   if (ShakeType == RTS_CTS)  {   // turn it off first, because it was
enabled
        setDTRoff();            // in the base class
        setRTSoff();
   }

   origcomint = getvect(comint);        // remember the original vector

   setvect(comint,COM3handler);            // point to the new handler

   setportbit(MCR,3);           // turn OUT2 on
   disable();           // disable all interrupts - critical section
   setportbit(IER,rx_rdy);      // enable ints on receive only
   origirq = inportb(IRQPORT);  // remember how it was
   clrportbit(IRQPORT,irqbit);  // enable COM ints

   if (ShakeType == RTS_CTS)  {
        setDTRon();
        setRTSon();
   }

   enable();

   EOI;
   cerr << "exiting crbPort constructor" << endl;
}
```

```
/***********************************************************************
   PROGRAM:   ~crbPortClass
   AUTHOR:    Kadir Akyol, Eric Bachmann
   DATE:      03 November 1998
   FUNCTION:  Resets the interrupts.
             1) disables the 8250 (async chip)
             2) disables the interrupt chip for async int
             3) resets the 8259 PIC
***********************************************************************/

crbPortClass::~crbPortClass()
{
   setvect(comint,origcomint);   // set the interrupt vector back
   outportb(IER,0);              // disable further UART interrupts
   outportb(MCR,0);              // turn everything off
   outportb(IRQPORT,origirq);
   EOI;
}

/***********************************************************************
   PROGRAM:   Get
   AUTHOR:    Kadir Akyol, Eric Bachmann
   DATE:      03 November 1998
   FUNCTION:  Calls Get based on buffer type
***********************************************************************/

Boolean crbPortClass::Get(CRBdata& data)
{
   return messages.Get(data);
}

/***********************************************************************
   PROGRAM: COM1handler
   AUTHOR:  Kadir Akyol, Eric Bachmann
   DATE:    11 July 1995, last modified November 1998
   FUNCTION:Specific interrupt handler which maps each interrupt to the
   proper ISR.
***********************************************************************/

void interrupt COM3handler(...)
{
   port3.processInterrupt();
   EOI;
}

/***********************************************************************
   PROGRAM:   processInterrupt
   AUTHOR:    Kadir Akyol, Eric Bachmann
   DATE:      03 November 1998
   FUNCTION:  Calls the ISR based upon buffer type
***********************************************************************/

void crbPortClass::processInterrupt()
{
   if (dataReady())  {             // make sure there's a char there
        BYTE data = inportb(RX);   // read character from 8250
        messages.Add(data);
        if (ShakeType == RTS_CTS && messages.capacityUsed() >
ALMOST_FULL)
             setDTRoff();
   }
}
// end of file crbport.cpp
```

P. SERIAL.H

```
#ifndef _SERIAL_H
#define _SERIAL_H

#include <dos.h>
#include <stdio.h>
#include "globals.h"

#define     ALMOST_FULL    80  // % full to turn off DTR (user defines)

// leave the following alone - hardware specific
enum  COMport           {COM1=1, COM2, COM3, COM4};
enum  BaudRate          {b300, b1200, b2400, b4800, b9600,b38400};
enum  ParityType        {ERROR=-1, NOPARITY, ODD, EVEN};
enum  handShake         {NONE, RTS_CTS, XON_XOFF};
enum  Shake             {off, on};
enum  interruptType     {rx_rdy, tx_rdy, line_stat, modem_stat};

#define  BIOSMEMSEG      0x40
#define  DLAB            0x80
#define  IRQPORT         0x21
#define  EOI             outportb(0x20,0x20)

#define  COM1base    MEMW(BIOSMEMSEG,0)
#define  COM2base    MEMW(BIOSMEMSEG,2)
#define  COM3base    MEMW(BIOSMEMSEG,4)

#define  TX                (portBase)
#define  RX                (portBase)
#define  IER               (portBase+1)
#define  IIR               (portBase+2)
#define  LCR               (portBase+3)
#define  MCR               (portBase+4)
#define  LSR               (portBase+5)
#define  MSR               (portBase+6)
#define  LO_LATCH          (portBase)
#define  HI_LATCH          (portBase+1)

/**********************************************************************
   CLASS:    serialPortClass
   AUTHOR:   Frank Kelbe,Eric Bachmann,Dave Gay,Rick Roberts,Kadir Akyol
   DATE:     11 July 1995, last modified March 1999
   FUNCTION:Parent class, defines a simple serial port.
**********************************************************************/

class serialPortClass  {

   public:

      serialPortClass(COMport port, BaudRate baud, ParityType parity,
                   BYTE wordlen, BYTE stopbits, handShake hs);
      ~serialPortClass() {}

      Boolean     Send(BYTE data);
      Boolean     Get(BYTE& data);
      inline Boolean dataReady();
      Boolean statusChanged()
       { return Boolean((ifportbit(MSR,0) || ifportbit(MSR,1))); }

       // the rest are only if handshake is specified as RTS_CTS
       Boolean     isCTSon()             { return ifportbit(MSR,4); }
       Boolean     isDSRon()             { return ifportbit(MSR,5); }

       void        setDTRon()            { setportbit(MCR,0); }
       void        setDTRoff()           { clrportbit(MCR,0); }
       void        toggleDTR();
       void        setRTSon()            { setportbit(MCR,1); }
       void        setRTSoff()           { clrportbit(MCR,1); }
       void        toggleRTS();

   protected:

      WORD           portBase;
      handShake      ShakeType;
      Shake             DTRstate, RTSstate;

      inline Boolean    ifportbit(WORD, BYTE);
      inline void           toggle(Shake&);

};

#endif
```

Q. SERIAL.CPP

```
#include <iostream.h>
#include <stdio.h>
#include "serial.h"

/*********************************************************************
   PROGRAM: serialPortClass (Constructor)
   AUTHOR:  Frank Kelbe,Eric Bachmann,Dave Gay,Rick Roberts,Kadir Akyol
   DATE:    11 July 1995, last modified March 1999
   FUNCTION:Initializes one of the Serial Ports.
         1) Determines the base I/O port address for the given COM port
         2) Sets the 8259 IRQ mask value
         3) Initializes the port parameters - baud, parity, etc.
         4) Calls the routine to initialize interrupt handling
         5) Enables DTR and RTS, indicating ready to go
*********************************************************************/

serialPortClass::serialPortClass(COMport port, BaudRate speed,
                            ParityType parity, BYTE wordlen,
                            BYTE stopbits, handShake hs)     :
                            DTRstate(off), RTSstate(off), ShakeType(hs)
{
   cerr << "serialPort constructor called" << endl;
   delay(500);

   switch (port)  {            // initialize appropriate port base
      case COM1:  portBase = COM1base;
      break;

      case COM2:  portBase = COM2base;
      break;

      case COM3:  portBase = COM3base;
      break;
   }  // switch

   const WORD     bauddiv[] = {0x180, 0x60, 0x30, 0x18, 0xC, 0x03};

   // Change 1
   outportb(IER,0);         // disable UART interrupts
   (void)inportb(LSR);
   (void)inportb(MSR);
   (void)inportb(IIR);
   (void)inportb(RX);

   outportb(LCR,DLAB);// set DLAB so can set baud rate (read only port)
   outportb(LO_LATCH,bauddiv[speed] & 0xFF);
   outportb(HI_LATCH,(bauddiv[speed] & 0xFF00) >> 8);
   setportbit(MCR,3);                          // turn OUT2 on

   BYTE  opt = 0;
   if (parity != NOPARITY)  {
        setbit(opt,3);      // enable parity
        if (parity == EVEN) // set even parity bit. if odd, leave bit 0
             setbit(opt,4);
   }
   // now set the word length. len of 5 sets both bits 0 and 1 to
   // 0, 6 sets to 01, 7 to 10 and 8 to 11
   opt |= wordlen-5;
   opt |= --stopbits << 2;
   outportb(LCR,opt);

   if (ShakeType = = RTS_CTS)  {
      setDTRon();
      setRTSon();
   }
   cerr << "serialPort constructed" << endl;
}

/**********************************************************************
   PROGRAM:   Get
   AUTHOR:    Frank Kelbe, Eric Bachmann, Dave Gay
   DATE:      11 July 1995
   FUNCTION:  Gets a byte from the port. Returns true if there's one
   there, and fills in the   byte parameter. If there's no character,
   the parameter is left alone, and false is returned.
**********************************************************************/

Boolean serialPortClass::Get(BYTE& data)
{
   if (dataReady())  {            // make sure there's a char there
      data = inportb(RX);         // read character from 8250
      return TRUE;
   }
   else
      return FALSE;
}

/**********************************************************************
   PROGRAM:   Send
   AUTHOR:    Frank Kelbe, Eric Bachmann, Dave Gay
   DATE:      11 July 1995
   FUNCTION:  Outputs a single character to the port. Returns Boolean
   status indicating  whether successful
**********************************************************************/

Boolean serialPortClass::Send(BYTE data)
{
   while (!(ifportbit(LSR,5))) {};        // wait until THR ready

   switch (ShakeType)  {
      case NONE:
           outportb(TX,data);
           return TRUE;

      case RTS_CTS:
      if (isCTSon() && isDSRon())  {
          outportb(TX,data);
          return TRUE;
      }
      else {
           return FALSE;
      }
```

```
     default:
          break;
   }

   return FALSE;
}

/**********************************************************************
   PROGRAM:  dataReady
   AUTHOR:   Frank Kelbe, Eric Bachmann, Dave Gay
   DATE:     11 July 1995
   FUNCTION: Checks port to see if a character has arrived.
**********************************************************************/

inline Boolean serialPortClass::dataReady()
{
/*    Un-commenting this code increases transmission errors, but this
      code is useful for troubleshooting, so is retained if needed
      if (ifportbit(LSR,1)) {
        cerr <<"\nOverrun Error\n";
      }
      if (ifportbit(LSR,2)) {
        cerr <<"\nParity Error\n";
      }
      if (ifportbit(LSR,3)) {
        cerr <<"\nFraming Error\n";
      }
*/
      return ifportbit(LSR,0);
}

/**********************************************************************
   PROGRAM:  ifportbit
   AUTHOR:   Frank Kelbe, Eric Bachmann, Dave Gay
   DATE:     11 July 1995
   FUNCTION: Checks for byte on inportb register
**********************************************************************/

inline Boolean serialPortClass::ifportbit(WORD reg, BYTE bit)
{
   BYTE on = inportb(reg);
   on &= set(bit);
   return Boolean(on == set(bit));
}

/**********************************************************************
   PROGRAM:  toggleDTR
   AUTHOR:   Frank Kelbe, Eric Bachmann, Dave Gay
   DATE:     11 July 1995
   FUNCTION: toggles Data Transmit Ready if RTS_CTS is off
**********************************************************************/

void serialPortClass::toggleDTR()
{
   if (ShakeType != RTS_CTS)
        return;
   if (DTRstate == off)
        setDTRon();
   else
        setDTRoff();
   toggle(DTRstate);
}

/*********************************************************************
   PROGRAM:  toggleRTS
   AUTHOR:   Frank Kelbe, Eric Bachmann, Dave Gay
   DATE:     11 July 1995
   FUNCTION: toggle Ready to Send (RTS) if RTS_CTS is on.
*********************************************************************/

void serialPortClass::toggleRTS()
{
   if (ShakeType != RTS_CTS)
       return;
   if (RTSstate == off)
       setRTSon();
   else
       setRTSoff();
   toggle(RTSstate);
}

/*********************************************************************
   PROGRAM:  toggle
   AUTHOR:   Frank Kelbe, Eric Bachmann, Dave Gay
   DATE:     11 July 1995
   FUNCTION: toggles value of the input variable
*********************************************************************/

inline void serialPortClass::toggle(Shake& h)
{
   if (h == off)
       h =  on;
   else
       h = off;
}
// end of file serial.cpp
```

LIST OF REFERENCES

1. Yuh, J., *Underwater Robotic Vehicle : Design and Control,* TSI Press, Albuquerque, New Mexico, 1995.

2. Brutzman, D.P., Burns, M., Campbell, M., Davis, D.T., Healey, A.J., Holden, M., Leonhardt, B.,Marco, D., McClarin, D., McGhee, R.B. and Whalen, R., "NPS Pheonix AUV Software Integration and In-Water Testing," *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology,* Monterey, California, June, 1996.

3. Hernandez, C.G., "An Integrated INS/GPS Navigation System for Small AUV Using an Asynchronous Kalman Filter," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1998.

4. Bachmann, E. R. and Gay, D., "Design and Evaluation of an Integrated GPS/INS System for Shallow-water AUV Navigation (SANS)," Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

5. McGhee, R.B., Clynch, J. R.., Healey, S. H., Kwak, S.H., Brutzman, D. P., Yun, X.P., Horton, N. A., Whalen, R. H., Bachamnn, E. R., Gay, D. L. and Shubert, W. R., "An Experimental Study of an Integrated GPS/INS System for Shallow-Water AUV Navigation (SANS)," *Proceedings of the Ninth International Symposium on Unmanned Untethered Submersible Technology (UUST)*, Durham, New Hampshire, September 25-27, 1995.

6. Knapp, R., "Design and Calibration of a Water Speed Sensor for a Small AUV Navigation System (SANS)," Master's Thesis, Naval Postgraduate School, Monterey, California, December 1997.

7. Walker, R. G., "Design and Evaluation of an Integrated, Self-contained GPS/INS Shallow-water AUV Navigation System (SANS)," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1996.

8. Roberts, R. L., "Experimental Evaluation, and Software Upgrade for Attitude Estimation by the Shallow-Water AUV Navigation System (SANS)," Master's Thesis, Naval Postgraduate School, Monterey, California, March 1997.

9. *TCM2 Electronic Compass Module User's Manual,* Precision Navigation Inc., June 1995.

10. *CMV586DX133 cpuModule User's Manual,* Real Time Devices, Inc., September 1997.

11. *CMT104 IDE Controller and Hard Drive Carrier utiliyModule,* Real Time Devices, Inc., September 1997.

12. *CM1122 Super VGA+Flat Panel utilityModule,* Real Time Devices, Inc., September 1997.

13. *C4-104 User Manual,* Sealevel Systems Inc., January 1997.

14. The Internet, [http://www.xbow.com/html/product.htm].

15. *Crossbow Six Axis Dynamic Measurement Unit Specifications,* Crossbow Technology, January 1998.

16. *Oncore User's Guide,* Motorola Inc., August 1995.

17. The Internet, [http://www.sontek.com/products/products.htm#HydraProd].

18. The Internet, [http://www.proxim.com/products/mobility/rl2/7400.shtml].

19. *DMU User's Manulal,* Crossbow Technology Inc., October 1998.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

4. Prof. Xiaoping Yun, Code EC/Yx 2
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

5. Eric Bachmann, Instructor, Code CS/Bc 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5118

6. Deniz Kuvvetleri Komutanligi 1
   Personel Tedarik ve Yetistirme Daire Baskanligi
   06100 Bakanliklar, Ankara
   TURKEY

7. Kadir Akyol 2
   Gunestepe Mah. Zuhre Sok. No:10/3
   34610 Gungoren, Istanbul
   TURKEY

8. Orta Dogu Teknik Universitesi 1
   Department of Computer Engineering
   06531 Ankara
   TURKEY

9. Bogazici Universitesi 1
   Department of Computer Engineering
   80815 Bebek, Istanbul
   TURKEY

10. Bilkent Universitesi 1
Department of Computer Engineering and Information Science
06533 Bilkent, Ankara
TURKEY

11. Deniz Harp Okulu Komutanligi 1
Kutuphane
Tuzla,Istanbul-81704,
TURKEY